Project Report

On

CLUSTERING GROCERY ITEMS

By

Name	College	Roll number
Rithika Akula	MVSR Engineering College	2451-18-733-025
Sai Sumanth Avara	MVSR Engineering College	2451-18-733-026
Ahmed Ata-ul Kareem	Sri Indu College of Engineering and Technology	17D41A0505
Bhimavarapu Indratej Reddy	Sphoorthy Engineering College	17N81A0552

(July 2020)

ABSTRACT

Online shops often sell tons of different items and this can become very messy very quickly! Data science can be extremely useful to automatically organize the products in categories so that they can be easily found by the customers.
The goal of this project is to look at user purchase history and create categories of items that are likely to be bought together and, therefore, should belong to the same section.
Company XYZ is an online grocery store. In the current version of the website, they have manually grouped the items into a few categories based on their experience.
However, they now have a lot of data about user purchase history. Therefore, they would like to put the data into use!
In this project we would like to answer the following questions:
The company founder wants to meet with some of the best customers to go through a focus group with them. We will identify the ID of the following customers to the founder:

- The customer who bought the most items overall in her lifetime
- For each item, the customer who bought that product the most
- Cluster items based on user co-purchase history. That is, create clusters of products that have the highest probability of being bought together. We will replace the old/manually created categories with these new ones. Each item can belong to just one cluster.

CONTENTS

INTRODUCTION

Company XYZ is an online grocery store. In the current version of the website, they have manually grouped the items into a few categories based on their experience.

Existing System:
The existing system does not provide a way of grouping customers and hence identifying natural clusters is difficult.
Disadvantages of Existing System:
The limitations of available systems are not sufficient to deal with the complex data. In this section, we present some of the limitations that are present in the existing system.

- The system uses DBMS and hence can return records based on the filters.
- The system also requires data extensive data preprocessing and Exploratory Data Analysis (EDA) in order to perform feature engineering.

However, they now have a lot of data about user purchase history. Therefore, they would like to put that data into use. The goal of this project is to look at this user purchase history and create categories of items that are likely to be bought together and, therefore, should belong to the same section.
We can use this manually prepared data and cluster items based on user co-purchase history. That is, create clusters of products that have the highest probability of being bought together. We will replace the old/manually created categories with these new ones. In this new system, each item can belong to just one cluster.

Proposed System:

We aim to implement K-Means, Hierarchical clustering and others and also fine tune the parameters of the model. These models would be trained on a data set which will be engineered carefully after performing the feature engineering.

Advantages:

- Load and explore the dataset and generate ideas for data preparation and model selection.
- Perform Exploratory Data Analysis to find correlations.
- Visualize clusters produced by the algorithms

The company founder wants to meet with some of the best customers to go through a focus group with them. So we will also identify the ID of the following customers to the founder:

- The customer who bought the most items overall in their lifetime
- For each item, the customer who bought that product the most

DATA COLLECTION

The .csv files given as dataset are:

- item_to_id.csv
- purchase_history.csv
- prepared_purchased_history.csv

***item_to_id.csv* file** contains two columns: *Item_name*, *Item_id*, that map the name of the grocery item to its assigned item id number.

***purchase_history.csv* file** contains two columns: *user_id*, *id*. The *user_id* column contains the id of the customer and its corresponding *id* column contains a list of item ids representing the items purchased by that customer. This file contains a raw representation of user purchase history.

***prepared_purchased_history.csv* file** contains 49 columns. The first column is the *id* of the customer. The following 48 columns (*item_1*, *item_2* … *item_48*) each represent an item from the list and the value of the cell corresponds to the quantity of that item bought by that customer. This file contains a clean representation of the user purchase history.

PROCEDURE

The following are the steps used to solve the problem statements.

- Importing Libraries
- Importing the dataset files
- Explore the data and pick the file relevant to the particular problem statement (since we have three of them)
- Perform exploratory Data Analysis/Visualization and bring insights of the variables to detect:
    a) The customer who bought the most items overall in their lifetime
    b) For each item, the customer who bought that product the most
- Prepare the data to perform Clustering efficiently
- Use Elbow method and/or Hierarchical Clustering to find optimum number of clusters
- Apply relevant Clustering algorithm to fit the clusters to the data
- Visualize the data
- Prepare the data to perform Classification efficiently
- Apply Logistic Regression Classifier algorithm to test the clusters created
- Measure the performance of the model using metrics like precision

DATA REPRESENTATION

item_to_id.csv file

First 5 rows of the dataset:

```
In [4]: item_to_id=pd.read_csv("item_to_id.csv")
        item_to_id.head()
```

Out[4]:

	Item_name	Item_id
0	coffee	43
1	tea	23
2	juice	38
3	soda	9
4	sandwich loaves	39

Shape:

```
In [112]: item_to_id.shape
Out[112]: (48, 2)
```

purchase_history.csv file

First 5 rows of the dataset:

```
In [5]: purchase_history=pd.read_csv("purchase_history.csv")
        purchase_history.head()
```

Out[5]:

	user_id	id
0	222087	27,26
1	1343649	6,47,17
2	404134	18,12,23,22,27,43,38,20,35,1
3	1110200	9,23,2,20,26,47,37
4	224107	31,18,5,13,1,21,48,16,26,2,44,32,20,37,42,35,4...

Shape:

```
In [113]: purchase_history.shape
Out[113]: (39474, 2)
```

prepared_purchased_history.csv file

There are total of 49 columns in our dataset the first one corresponds to the customer id and the rest representing the 48 grocery items available.

First 5 rows of the dataset:

```
In [8]: prepared_purchased_history=pd.read_csv("prepared_purchased_history.csv")
        prepared_purchased_history.head()
```

Out[8]:

	id	item_1	item_2	item_3	item_4	item_5	item_6	item_7	item_8	item_9	...	item_39	item
0	47	0	1	1	1	0	0	0	0	0	...	0	
1	68	0	0	0	0	0	1	0	0	0	...	1	
2	113	0	0	1	0	0	0	0	0	1	...	0	
3	123	0	0	0	1	0	0	0	0	0	...	0	
4	223	1	1	0	0	0	1	0	0	0	...	0	

5 rows × 49 columns

(Continued…)

Out[8]:

n_9	...	item_39	item_40	item_41	item_42	item_43	item_44	item_45	item_46	item_47	item_48
0	...	0	0	0	0	0	1	1	1	0	0
0	...	1	0	0	1	0	0	0	0	0	0
1	...	0	0	0	0	1	0	0	1	0	0
0	...	0	0	0	0	0	0	0	0	0	0
0	...	0	0	1	0	0	0	1	0	0	0

Shape:

```
In [114]: prepared_purchased_history.shape
Out[114]: (24885, 49)
```

DATA ANALYSIS

We have checked for categorical feature and found that there are none. So we have concluded that there are only numerical features in the *prepared_purchased_history* file.

Now we can begin working on our problem statements.

1. Detect the customer who bought the most items overall in their lifetime

Importing Libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

Importing Dataset files

```
In [10]: item = pd.read_csv("item_to_id.csv")
         purchase = pd.read_csv("purchase_history.csv")
```

Creating Item Dictionary

```
In [13]: item_dict = item.set_index('Item_id').to_dict()['Item_name']
         print(item_dict)

         {43: 'coffee', 23: 'tea', 38: 'juice', 9: 'soda', 39: 'sandwich lo
         aves', 37: 'dinner rolls', 34: 'tortillas', 13: 'bagels', 28: 'can
         ned vegetables', 26: 'spaghetti sauce', 41: 'ketchup', 21: 'cheese
         s', 14: 'eggs', 16: 'milk', 48: 'yogurt', 8: 'butter', 11: 'cereal
         s', 30: 'flour', 1: 'sugar', 31: 'pasta', 5: 'waffles', 22: 'froze
         n vegetables', 36: 'ice cream', 6: 'poultry', 17: 'beef', 47: 'por
         k', 46: 'bananas', 40: 'berries', 25: 'cherries', 20: 'grapefrui
         t', 32: 'apples', 44: 'broccoli', 10: 'carrots', 45: 'cauliflowe
         r', 42: 'cucumbers', 2: 'lettuce', 18: 'laundry detergent', 27: 'd
         ishwashing\xa0', 24: 'paper towels', 33: 'toilet paper', 15: 'alum
         inum foil', 7: 'sandwich bags', 12: 'shampoo', 35: 'soap', 29: 'ha
         nd soap', 19: 'shaving cream', 4: 'baby items', 3: 'pet items'}
```

Creating new column in *purchase* by name *items_bought*

```
In [14]: purchase["items_bought"]=0
```

Counting the number of items in *id* column

```
In [15]: total_items_purchased = []
         for i in range(len(purchase)):
             total_items_purchased.append(len(purchase["id"].loc[i].split(",")) )
```

Assigning the count to corresponding *items_bought* column value

```
In [16]: purchase["items_bought"]=total_items_purchased
```

Arranging user_ids in ascending order of item bought

```
In [21]: df = purchase[['user_id','items_bought']]
         df_grpmean=df.groupby('user_id').sum()
```

Printing the top customer who bought the most items overall in their lifetime

```
In [22]: df_grpmean.sort_values(['items_bought'], ascending=False).head(1)
```

Out[22]:

	items_bought
user_id	
269335	72

The user with user_id = 269335 has bought 72 items in their lifetime.

2. For each item, find the customer who bought that product the most

Creating *item_arr* of purchase items (shopping-cart) per transaction for 39474 transactions

```
In [23]: item_arr = []
         for i in range(len(purchase)):
             item_arr.append(purchase["id"].loc[i].split(","))
```

Mapping shopping-cart *item_ids* to *item_name*

```
In [24]: final_arr = []
         for i in range(len(item_arr)):
             temp_arr = []
             temp_arr = item_arr[i]
             for i in range(len(temp_arr)):
                 temp_arr[i]  = item_dict[int(temp_arr[i])]
             final_arr.append(temp_arr)
```

Creating array with *user_id* values

```
In [25]: purchase_user_arr = purchase['user_id']
```

Mapping *customer_id* with purchased grocery items

```
In [26]: user_to_item =[]
         for i in range(len(final_arr)):
             temp_arr = []
             temp_arr = final_arr[i]
             for j in range(len(temp_arr)):
                    user_to_item.append([purchase_user_arr[i], temp_arr[j]])
```

Creating *user_id* to *purchased item* DataFrame

```
In [27]: user_item_df = pd.DataFrame(user_to_item, columns=['user_id', 'item_name'])
         user_item_df.head(5)
```

Out[27]:

	user_id	item_name
0	222087	dishwashing
1	222087	spaghetti sauce
2	1343649	poultry
3	1343649	pork
4	1343649	beef

Identifying unique items and counting number of times it was purchased

```
In [28]: item_list = user_item_df['item_name'].unique()
```

```
In [29]: item_fav_customer = []
         for item in item_list:
             item_df = user_item_df[user_item_df["item_name"] == item]
             item_fav_customer.append([item, item_df['user_id'].value_counts().idxmax()])
```

We now create a DataFrame '*df*' to store the *item name* and the *customer id* of the customer who has bought that item the most.

```
In [111]: df = pd.DataFrame(item_fav_customer)
          df.columns = ['item', 'customer id']
          df
```

df :

	item	customer id
0	dishwashing	956666
1	spaghetti sauce	1341188
2	poultry	334664
3	pork	1374100
4	beef	366155
5	laundry detergent	917199
6	shampoo	791038
7	tea	920002
8	frozen vegetables	1199670
9	coffee	996380
10	juice	255546
11	grapefruit	1433799
12	soap	1003550
13	sugar	1301034
14	soda	397623
15	lettuce	31625
16	dinner rolls	364868
17	pasta	289360
18	waffles	217277
19	bagels	820788
20	cheeses	884172
21	yogurt	943163
22	milk	837807
23	broccoli	297185
24	apples	1303742
25	cucumbers	80215
26	berries	384935
27	sandwich bags	360336
28	hand soap	394348
29	butter	478446
30	cauliflower	1198106
31	aluminum foil	143741
32	cereals	367872
33	cherries	109578
34	eggs	172120
35	ketchup	133355
36	canned vegetables	238495
37	carrots	743501
38	bananas	1218645
39	pet items	1433188
40	shaving cream	31625
41	sandwich loaves	599172
42	flour	1076958
43	tortillas	1485538
44	toilet paper	1425746
45	paper towels	1077463
46	ice cream	269335
47	baby items	73071

DATA PREPARATION

Algorithms require features with some specific characteristic to work properly. Here, the need for **Feature Engineering** arises. Feature engineering efforts mainly have two goals:

- Preparing the proper input dataset, compatible with the machine learning algorithm requirements.
- Improving the performance of machine learning models.

In our dataset *prepared_purchased_history.csv* file

```
In [2]: dataset=pd.read_csv("prepared_purchased_history.csv")
```

```
In [3]: dataset.describe()
```

Out[3]:

	id	item_1	item_2	item_3	item_4	item_5	item_6	item_7	item_8	item_9	...	
count	2.488500e+04	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	...	2488
mean	7.508893e+05	0.366446	0.581595	0.289492	0.131083	0.113201	0.350814	0.134338	0.229737	0.354270	...	
std	4.336508e+05	0.562783	0.679563	0.508112	0.353889	0.330988	0.553713	0.358599	0.459569	0.555357	...	
min	4.700000e+01	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	3.737880e+05	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	7.512480e+05	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	1.122789e+06	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	...	
max	1.499974e+06	4.000000	5.000000	4.000000	3.000000	3.000000	4.000000	3.000000	3.000000	4.000000	...	

8 rows × 49 columns

(Continued…)

Out[3]:

item_9	...	item_39	item_40	item_41	item_42	item_43	item_44	item_45	item_46	item_47	item_48
24885.000000	...	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000	24885.000000
0.354270	...	0.350733	0.356761	0.257987	0.363673	0.352984	0.360860	0.357726	0.358489	0.348885	0.227969
0.555357	...	0.558389	0.558960	0.482524	0.560543	0.552769	0.563714	0.555312	0.556014	0.552609	0.454298
0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	...	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
4.000000	...	5.000000	4.000000	4.000000	4.000000	4.000000	4.000000	5.000000	4.000000	4.000000	3.000000

We have the first column representing the customer *id*. For forming clusters of the grocery items, we do not need the *id* column and hence this column needs to be removed.

In [10]:
```
dataset.drop(["id"], axis=1, inplace=True)
dataset
```

Out[10]:

	item_1	item_2	item_3	item_4	item_5	item_6	item_7	item_8	item_9	item_10	...	item_39	item_40	item_41	item_4
0	0	1	1	1	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	1	0	0	0	1	...	1	0	0	
2	0	0	1	0	0	0	0	0	1	0	...	0	0	0	
3	0	0	0	1	0	0	0	0	0	1	...	0	0	0	
4	1	1	0	0	0	1	0	0	0	0	...	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24880	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
24881	0	1	0	0	0	1	0	0	0	0	...	0	0	0	
24882	0	0	1	0	0	1	0	0	0	0	...	1	0	0	
24883	1	2	0	0	0	1	0	1	1	2	...	0	0	0	
24884	0	0	0	0	1	0	1	1	0	1	...	1	0	0	

24885 rows × 48 columns

(Continued…)

Out[10]:

item_7	item_8	item_9	item_10	...	item_39	item_40	item_41	item_42	item_43	item_44	item_45	item_46	item_47	item_48
0	0	0	0	...	0	0	0	0	0	1	1	1	0	0
0	0	0	1	...	1	0	0	1	0	0	0	0	0	0
0	0	1	0	...	0	0	0	0	1	0	0	1	0	0
0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
0	0	0	0	...	0	0	1	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	0	0	...	0	0	0	0	0	0	1	0	0	0
0	0	0	0	...	0	0	0	1	0	0	0	0	0	1
0	0	0	0	...	1	0	0	0	0	1	1	0	0	1
0	1	1	2	...	0	0	0	2	1	2	1	0	1	1
1	1	0	1	...	1	0	0	0	0	0	0	0	0	0

According to the problem statement, we need whether an item was bought by the customer or not. We do not need the quantity of the items bought. Hence we need a table representing value '1' for item bought and '0' for not bought.

Using the *user_item_df* DataFrame created previously,

```
In [25]: user_item_df.head()
```

Out[25]:

	user_id	item_name
0	222087	dishwashing
1	222087	spaghetti sauce
2	1343649	poultry
3	1343649	pork
4	1343649	beef

Adding a new column *bought* to *user_item_df*,

```
In [26]: user_item_df['bought']=1
```

```
In [27]: user_item_df.head()
```

Out[27]:

	user_id	item_name	bought
0	222087	dishwashing	1
1	222087	spaghetti sauce	1
2	1343649	poultry	1
3	1343649	pork	1
4	1343649	beef	1

Using the above DataFrame with *user_id* and *item_name*, we create a table *useritem_pivot* containing value '1' for item purchased by that user and NaN for not purchased items.

```
In [28]: useritem_pivot = user_item_df.pivot_table(index=['user_id'],columns=['item_name'],values='bought')
         useritem_pivot.head(5)
```

Out[28]:

item_name	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables
user_id										
47	1.0	NaN	1.0	1.0	1.0	NaN	NaN	1.0	NaN	1.0
68	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
113	1.0	1.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
123	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
223	NaN	1.0	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN

5 rows × 48 columns

(Continued…)

Out[28]:

utter	canned vegetables	...	shaving cream	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt
NaN	1.0	...	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
NaN	NaN	...	1.0	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
NaN	NaN	...	NaN	NaN	1.0	NaN	NaN	NaN	1.0	NaN	NaN	NaN
NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	...	NaN	1.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN

Filling all NaN valued cells with ‘0’,

```
In [30]: useritem_pivot.fillna(0, inplace=True)
         useritem_pivot.head()
```

Out[30]:

item_name user_id	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables
47	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0
68	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
113	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
123	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
223	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 48 columns

(Continued…)

Out[30]:

butter	canned vegetables	...	shaving cream	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt
0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Saving the *useritem_pivot* table to csv file *useritem_pivot.csv*,

```
In [38]: useritem_pivot.to_csv("useritem_pivot.csv")
         useritem = pd.read_csv("useritem_pivot.csv")
```

Dropping the *user_id* column,

```
In [39]: useritem.drop(['user_id'], axis=1, inplace=True)
         useritem.head()
```

Out[39]:

	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables	...	sha cr
0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	...	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	
2	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
4	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	

5 rows × 48 columns

(Continued…)

Out[39]:

butter	canned vegetables	...	shaving cream	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt
0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

CLUSTERING

Now we can begin clustering our data.
Elbow Method is used to determine the optimal number of clusters possible for the data.

<u>Elbow Method:</u>

Importing KMeans library from sklearn.cluster

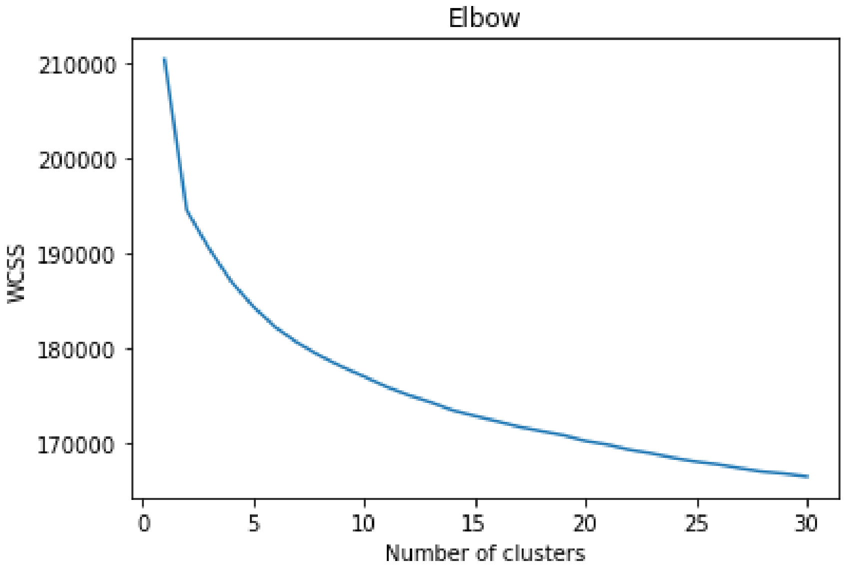
```
In [40]: from sklearn.cluster import KMeans
```

x is a numpy array made from *useritem* DataFrame

```
In [42]: x=useritem.values
```

Plotting the elbow curve by varying cluster size vs. squared error

```
In [43]: wcss=[]
         for i in range(1,31):
             kmeans=KMeans(n_clusters=i,init='k-means++',random_state=42)
             kmeans.fit(x)
             wcss.append(kmeans.inertia_)
         plt.plot(range(1,31),wcss)
         plt.title('Elbow')
         plt.xlabel('Number of clusters')
         plt.ylabel('WCSS')
         plt.show()
```



No sharp elbow detected.

Let us try another method of clustering, Hierarchical Clustering because:

- The hierarchical clustering algorithm determines the optimal number of clusters by plotting a dendrogram.
- We can visualize the categories of items and see which items fall in the same cluster effectively in hierarchical clustering.

Hierarchical Clustering:

Preparation of Data for Hierarchical Clustering:

In order to perform hierarchical clustering, the data should be in the form of a correlation matrix.

Creating *corr*, correlation matrix of shape (48 x 48) for *useritem* DataFrame

```
In [47]: corr = useritem.corr()
         corr.head()
```

Out[47]:

	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables	...	shaving cream
aluminum foil	1.000000	0.061678	0.047728	0.064905	0.067114	0.071278	0.055401	0.072599	0.071912	0.065412	...	0.045270
apples	0.061678	1.000000	0.062212	0.087139	0.225915	0.095325	0.223348	0.087706	0.084418	0.077379	...	0.070087
baby items	0.047728	0.062212	1.000000	0.071940	0.074922	0.070992	0.076462	0.063789	0.064575	0.048423	...	0.050868
bagels	0.064905	0.087139	0.071940	1.000000	0.077304	0.087987	0.093777	0.084959	0.092361	0.082148	...	0.075134
bananas	0.067114	0.225915	0.074922	0.077304	1.000000	0.082479	0.219283	0.093150	0.076063	0.084400	...	0.069899

5 rows × 48 columns

(Continued...)

Out[47]:

i	butter	canned vegetables	...	shaving cream	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt
)	0.071912	0.065412	...	0.045270	0.036808	0.071673	0.059981	0.070864	0.062613	0.133057	0.070844	0.034712	0.048195
5	0.084418	0.077379	...	0.070087	0.065698	0.089582	0.079165	0.102892	0.083509	0.070139	0.086649	0.055191	0.074820
)	0.064575	0.048423	...	0.050868	0.048610	0.058916	0.045073	0.118216	0.064678	0.058797	0.068229	0.053431	0.063001
)	0.092361	0.082148	...	0.075134	0.066404	0.091899	0.079857	0.097317	0.091047	0.060387	0.251497	0.051933	0.087520
)	0.076063	0.084400	...	0.069899	0.067265	0.092232	0.082774	0.102371	0.091121	0.072508	0.094638	0.065618	0.073154

Now, we create a list containing strings representing the item name in order to use as labels for the leaf nodes in the dendrogram.

Using the *item_name* column from *user_item_df* DataFrame:

In [39]:
```
user_item_df.head()
```

Out[39]:

	user_id	item_name	bought
0	222087	dishwashing	1
1	222087	spaghetti sauce	1
2	1343649	poultry	1
3	1343649	pork	1
4	1343649	beef	1

Extracting the unique values of *item_name* column and sorting, and storing them in a list named *items*

```
In [38]: items=user_item_df['item_name'].unique()
         items=sorted(items)
         print(items)
```

```
['aluminum foil', 'apples', 'baby items', 'bagels', 'bananas', 'beef', 'berrie
s', 'broccoli', 'butter', 'canned vegetables', 'carrots', 'cauliflower', 'cerea
ls', 'cheeses', 'cherries', 'coffee', 'cucumbers', 'dinner rolls', 'dishwashing
\xa0', 'eggs', 'flour', 'frozen vegetables', 'grapefruit', 'hand soap', 'ice cr
eam', 'juice', 'ketchup', 'laundry detergent', 'lettuce', 'milk', 'paper towel
s', 'pasta', 'pet items', 'pork', 'poultry', 'sandwich bags', 'sandwich loave
s', 'shampoo', 'shaving cream', 'soap', 'soda', 'spaghetti sauce', 'sugar', 'te
a', 'toilet paper', 'tortillas', 'waffles', 'yogurt']
```

Converting *corr* of type DataFrame, to numpy array *x*:

```
In [37]: x=corr.values
         x
```

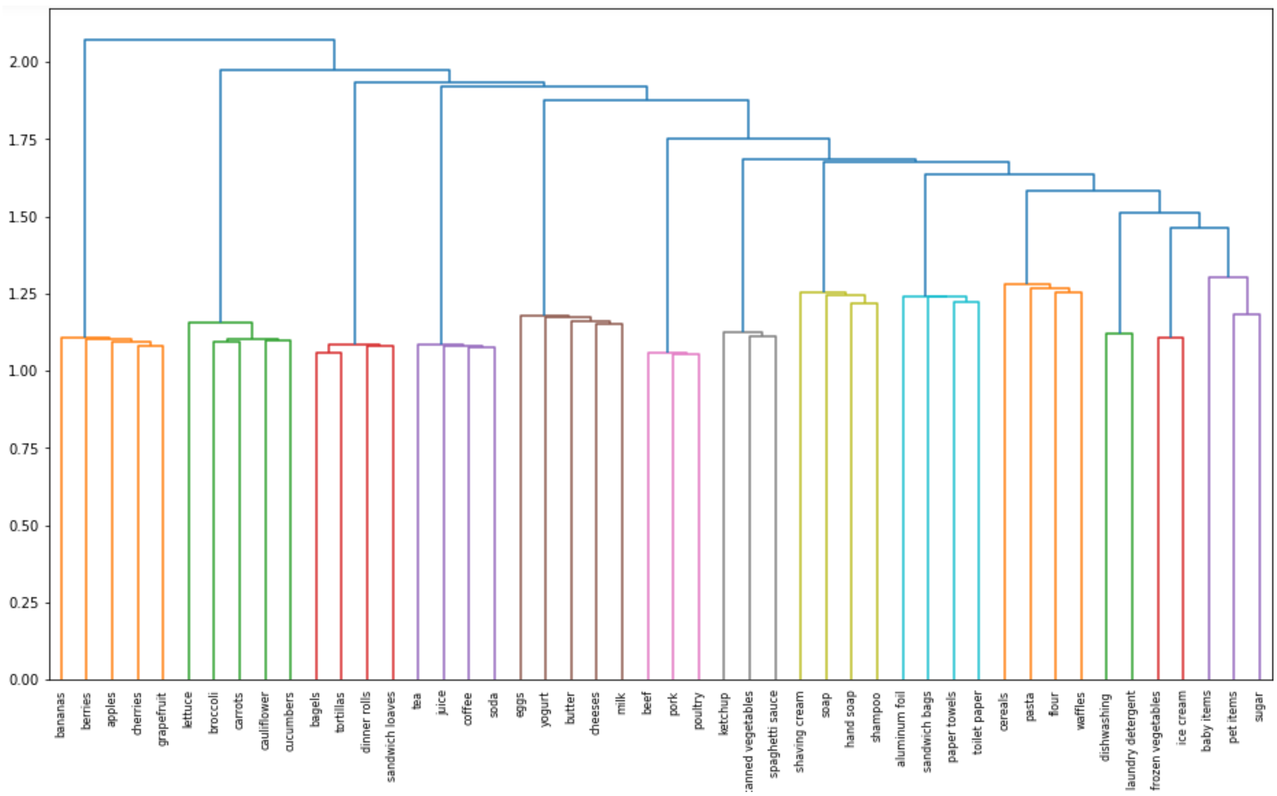
```
Out[37]: array([[1.        , 0.06167769, 0.04772773, ..., 0.07084396, 0.03471213,
                 0.04819532],
                [0.06167769, 1.        , 0.06221164, ..., 0.08664868, 0.05519078,
                 0.07482042],
                [0.04772773, 0.06221164, 1.        , ..., 0.06822855, 0.05343142,
                 0.06300134],
                ...,
                [0.07084396, 0.08664868, 0.06822855, ..., 1.        , 0.05518335,
                 0.08467477],
                [0.03471213, 0.05519078, 0.05343142, ..., 0.05518335, 1.        ,
                 0.04653839],
                [0.04819532, 0.07482042, 0.06300134, ..., 0.08467477, 0.04653839,
                 1.        ]])
```

Plotting the dendrogram:

Importing scipy.cluster.hierarchy library:

```
In [40]: import scipy.cluster.hierarchy as sch
```

```
In [52]: fig, ax = plt.subplots(figsize = (16, 9))
         dendrogram = sch.dendrogram(sch.linkage(x, method="ward"), ax=ax, labels=items)
```



Creating a list representing the cluster number for each item:

Importing *fcluster* from *scipy.cluster.hierarchy*:

```
In [ ]: from scipy.cluster.hierarchy import fcluster
```

Passing the number of clusters to be 14, according to the dendrogram above

```
In [60]: fl = fcluster(sch.linkage(x, method = "ward"), 14, criterion = 'maxclust')
         fl
```

The array *fl* contains the cluster number corresponding to the item in the items list.

```
Out[60]: array([ 9,  1, 14,  3,  1,  6,  1,  2,  5,  7,  2,  2, 10,  5,  1,  4,  2,
                 3, 11,  5, 10, 12,  1,  8, 12,  4,  7, 11,  2,  5,  9, 10, 13,  6,
                 6,  9,  3,  8,  8,  8,  4,  7, 13,  4,  9,  3, 10,  5],
               dtype=int32)
```

Here is the *items* list created previously:

```
['aluminum foil', 'apples', 'baby items', 'bagels', 'bananas', 'beef', 'berrie
s', 'broccoli', 'butter', 'canned vegetables', 'carrots', 'cauliflower', 'cerea
ls', 'cheeses', 'cherries', 'coffee', 'cucumbers', 'dinner rolls', 'dishwashing
\xa0', 'eggs', 'flour', 'frozen vegetables', 'grapefruit', 'hand soap', 'ice cr
eam', 'juice', 'ketchup', 'laundry detergent', 'lettuce', 'milk', 'paper towel
s', 'pasta', 'pet items', 'pork', 'poultry', 'sandwich bags', 'sandwich loave
s', 'shampoo', 'shaving cream', 'soap', 'soda', 'spaghetti sauce', 'sugar', 'te
a', 'toilet paper', 'tortillas', 'waffles', 'yogurt']
```

```
In [77]: result=list(zip(fl,items))
         result=sorted(result)
         print(result)
```

```
[(1, 'apples'), (1, 'bananas'), (1, 'berries'), (1, 'cherries'), (1, 'grapefrui
t'), (2, 'broccoli'), (2, 'carrots'), (2, 'cauliflower'), (2, 'cucumbers'), (2,
'lettuce'), (3, 'bagels'), (3, 'dinner rolls'), (3, 'sandwich loaves'), (3, 'to
rtillas'), (4, 'coffee'), (4, 'juice'), (4, 'soda'), (4, 'tea'), (5, 'butter'),
(5, 'cheeses'), (5, 'eggs'), (5, 'milk'), (5, 'yogurt'), (6, 'beef'), (6, 'por
k'), (6, 'poultry'), (7, 'canned vegetables'), (7, 'ketchup'), (7, 'spaghetti s
auce'), (8, 'hand soap'), (8, 'shampoo'), (8, 'shaving cream'), (8, 'soap'),
(9, 'aluminum foil'), (9, 'paper towels'), (9, 'sandwich bags'), (9, 'toilet pa
per'), (10, 'cereals'), (10, 'flour'), (10, 'pasta'), (10, 'waffles'), (11, 'di
shwashing\xa0'), (11, 'laundry detergent'), (12, 'frozen vegetables'), (12, 'ic
e cream'), (13, 'pet items'), (13, 'sugar'), (14, 'baby items')]
```

Categories of items likely to be bought together:

Cluster 1	Apples, Bananas, Berries, Cherries, Grapefruit
Cluster 2	Broccoli, Carrots, Cauliflower, Cucumber, Lettuce
Cluster 3	Bagels, Dinner Rolls, Sandwich Loaves, Tortillas
Cluster 4	Coffee, Juice, Soda, Tea
Cluster 5	Butter, Cheeses, Eggs, Milk, Yoghurt
Cluster 6	Beef, Pork, Poultry
Cluster 7	Canned Vegetables, Ketchup, Spaghetti Sauce
Cluster 8	Hand Soap, Shampoo, Shaving Cream, Soap
Cluster 9	Aluminum Foil, Paper Towels, Sandwich Bags, Toilet Paper
Cluster 10	Cereals, Flour, Pasta, Waffles
Cluster 11	Dishwashing, Laundry Detergent
Cluster 12	Frozen Vegetables, Ice Cream
Cluster 13	Pet items, Sugar
Cluster 14	Baby items

Fitting the clusters to our data:

We have found the optimal number of clusters, now we need to fit this to our data.

```
In [45]: x=useritem.values
         x
```

```
Out[45]: array([[1., 0., 1., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [1., 1., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 1., 0., 1.],
                [0., 1., 0., ..., 1., 0., 1.],
                [0., 0., 0., ..., 1., 1., 0.]])
```

Using **Agglomerative Clustering** to fit the formed clusters to the data *x*:

```
In [ ]: from sklearn.cluster import AgglomerativeClustering

        cluster = AgglomerativeClustering(n_clusters=14, affinity='euclidean', linkage='ward')
        cluster.fit_predict(x)
```

The above code throws a **MemoryError**.

```
MemoryError: unable to allocate array data.
```

Fitting the clusters using any algorithm other than KMeans Clustering throws a Memory Error. Hence we need to use **KMeans Clustering** for this data.

K Means Clustering:

Preparation of Data for KMeans Clustering:

Principal component analysis (PCA). Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

```
In [8]: from sklearn.decomposition import PCA
        pca=PCA(n_components=2)
        x=pca.fit_transform(x)
```

(KMeans imported previously for Elbow Method)

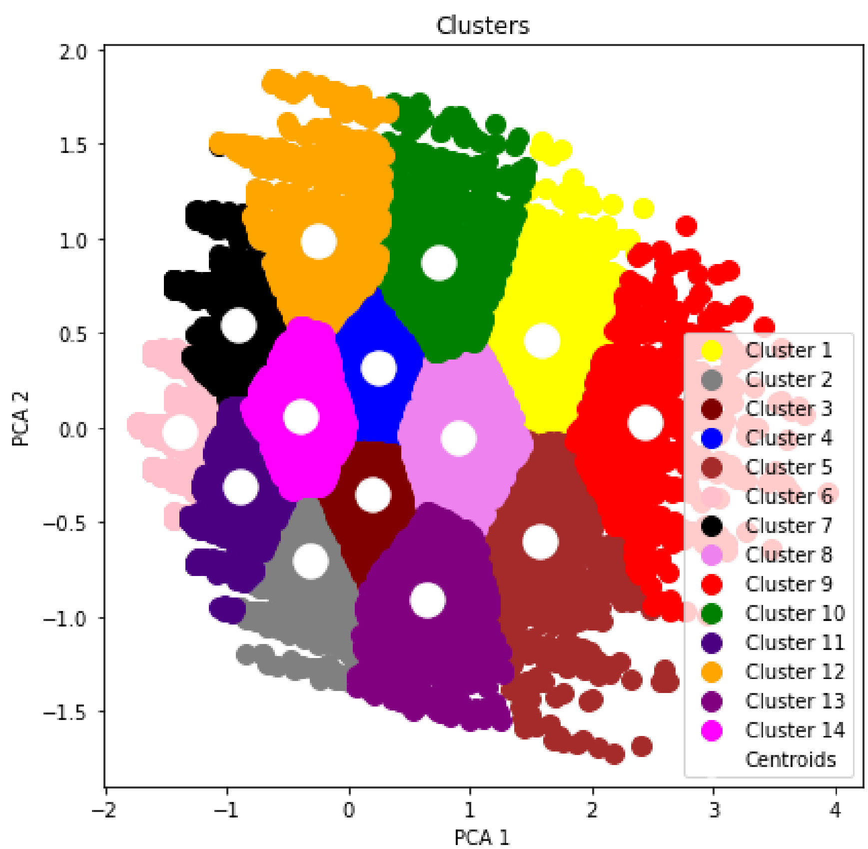
Number of clusters chosen = 14

```
In [47]: kmeans = KMeans(n_clusters= 14,init = 'k-means++', random_state = 0)
         y_kmeans = kmeans.fit_predict(x)
```

Plotting the data with formed clusters:

```
In [60]: plt.figure(figsize=(7,7))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'yellow', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'gray', label = 'Cluster 2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'maroon', label = 'Cluster 3')
plt.scatter(x[y_kmeans == 3, 0], x[y_kmeans == 3, 1], s = 100, c = 'blue', label = 'Cluster 4')
plt.scatter(x[y_kmeans == 4, 0], x[y_kmeans == 4, 1], s = 100, c = 'brown', label = 'Cluster 5')
plt.scatter(x[y_kmeans == 5, 0], x[y_kmeans == 5, 1], s = 100, c = 'pink', label = 'Cluster 6')
plt.scatter(x[y_kmeans == 6, 0], x[y_kmeans == 6, 1], s = 100, c = 'black', label = 'Cluster 7')
plt.scatter(x[y_kmeans == 7, 0], x[y_kmeans == 7, 1], s = 100, c = 'violet', label = 'Cluster 8')
plt.scatter(x[y_kmeans == 8, 0], x[y_kmeans == 8, 1], s = 100, c = 'red', label = 'Cluster 9')
plt.scatter(x[y_kmeans == 9, 0], x[y_kmeans == 9, 1], s = 100, c = 'green', label = 'Cluster 10')
plt.scatter(x[y_kmeans == 10,0], x[y_kmeans == 10,1], s = 100, c = 'indigo', label = 'Cluster 11')
plt.scatter(x[y_kmeans == 11,0], x[y_kmeans == 11,1], s = 100, c = 'orange', label = 'Cluster 12')
plt.scatter(x[y_kmeans == 12,0], x[y_kmeans == 12,1], s = 100, c = 'purple', label = 'Cluster 13')
plt.scatter(x[y_kmeans == 13,0], x[y_kmeans == 13,1], s = 100, c = 'magenta', label = 'Cluster 14')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'white',
            label = 'Centroids')
plt.title('Clusters')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.show()
```

Scatter plot:



DATA PREPARATION

Now that we have formed the optimal number of clusters for our data, we need to combine the dataset with clusters predicted to get accuracy score of the classification.

Copying *useritem* DataFrame to *dataset*:

```
In [61]: dataset=useritem.copy()
         dataset
```

Out[61]:

	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables	...	shaving cream	soap	soda	spagh sa
0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	
2	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
24880	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
24881	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
24882	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	
24883	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	...	1.0	0.0	1.0	
24884	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	

24885 rows × 48 columns

(Continued…)

Out[61]:

as	beef	berries	broccoli	butter	canned vegetables	...	shaving cream	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt
1.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
0.0	1.0	0.0	1.0	1.0	0.0	...	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0

Reset the index of the DataFrame, and use the default one instead.

```
In [62]: dataset=dataset.reset_index()
```

Creating DataFrame *clustersdf*

```
In [63]: clustersdf = pd.DataFrame(y_kmeans)
         clustersdf.columns = ['cluster_predicted']
```

Creating *combinedDF* that contains *dataset* and *cluster_predicted* columns

```
In [79]: combinedDF = pd.concat([dataset, clustersdf], axis = 1).reset_index()

         combinedDF.drop(['index','level_0'],axis=1,inplace=True)
         combinedDF.head()
```

Out[79]:

	aluminum foil	apples	baby items	bagels	bananas	beef	berries	broccoli	butter	canned vegetables	...	soap	soda	spaghetti sauce	sugar	t
0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1
2	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0
4	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	1.0	0

5 rows × 49 columns

(Continued…)

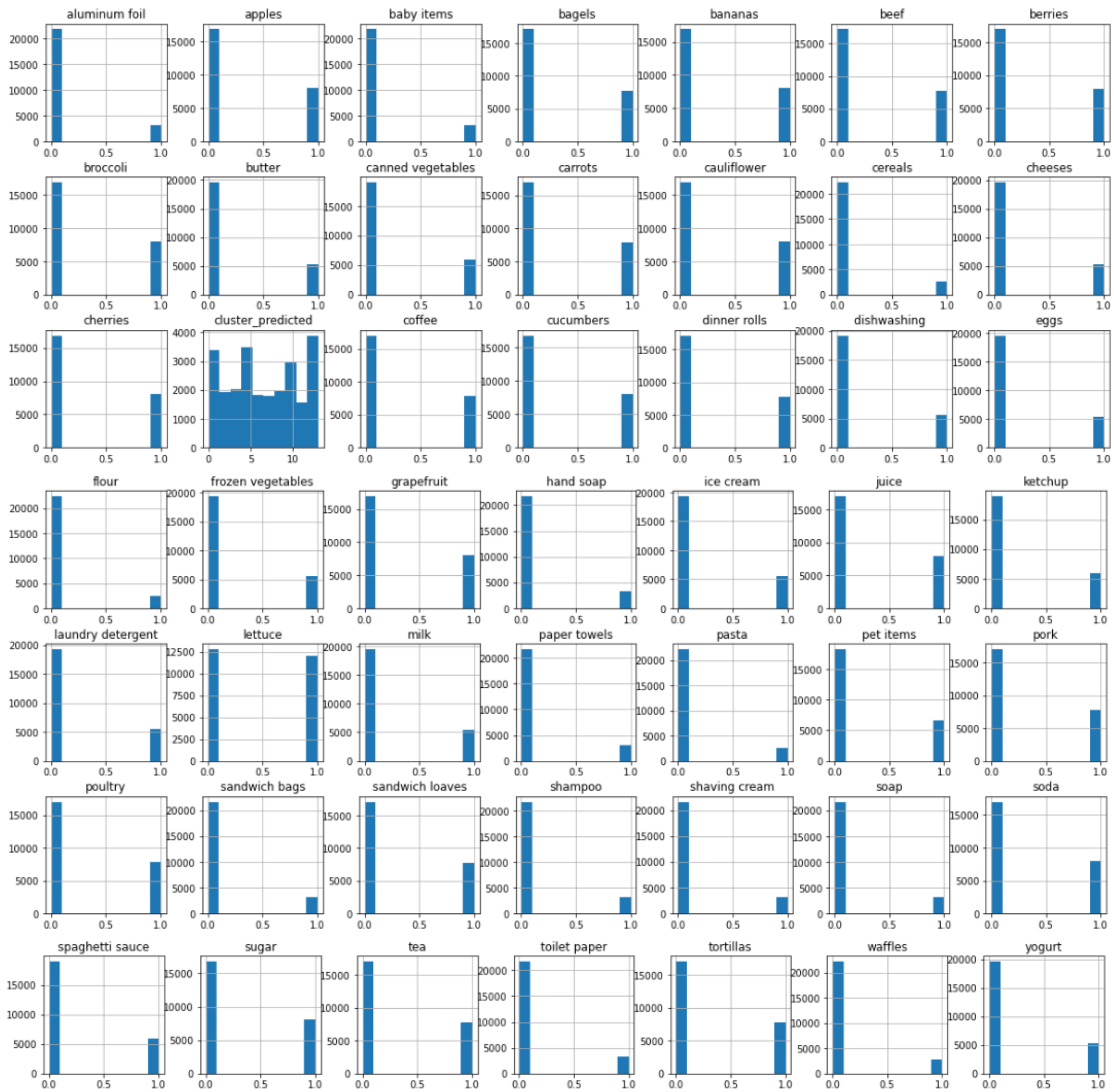
Out[79]:

berries	broccoli	butter	canned vegetables	...	soap	soda	spaghetti sauce	sugar	tea	toilet paper	tortillas	waffles	yogurt	cluster_predicted
0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	2
0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	13
0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	11
0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5
0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	2

DATA VISUALIZATION

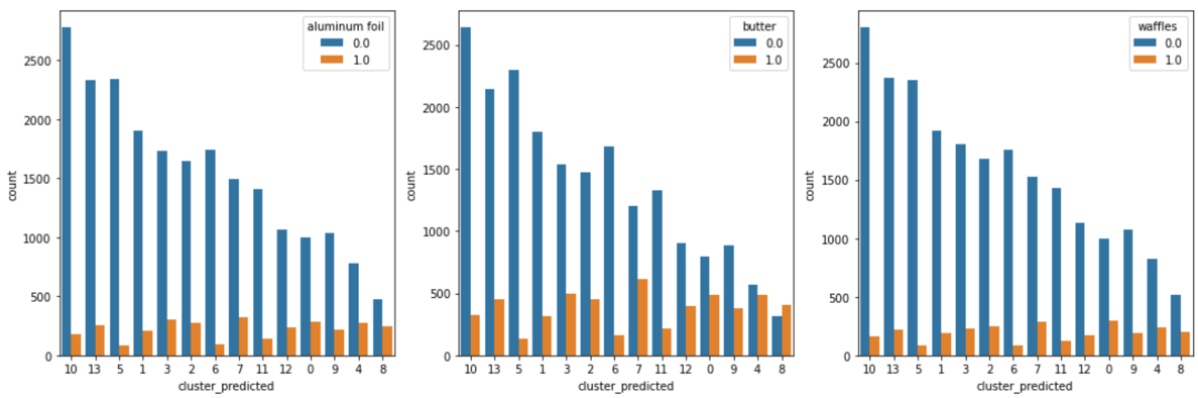
Plotting Histograms for all columns in *combinedDF* dataset:

```
In [74]: combinedDF.hist(figsize=(20,20))
         plt.show()
```



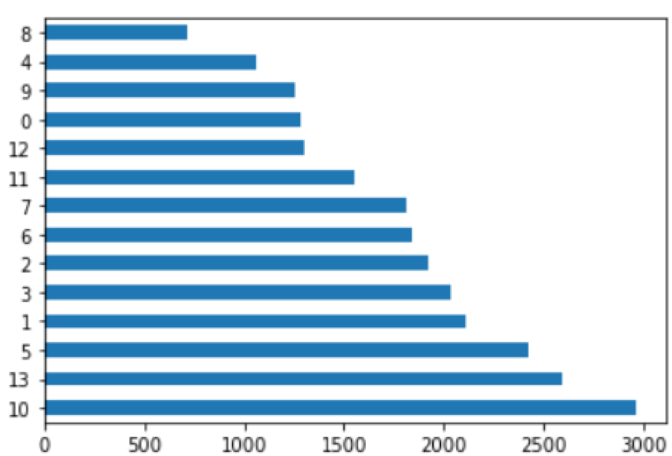
Plotting countplots for 3 columns, *aluminum foil*, *butter* and *waffles*:

```
In [69]: f, axs = plt.subplots(1,3,figsize = (15,5))
         sns.countplot(x=combinedDF['cluster_predicted'],
                       order=combinedDF['cluster_predicted'].value_counts().index,
                       hue=combinedDF['aluminum foil'],ax=axs[0])
         sns.countplot(x=combinedDF['cluster_predicted'],
                       order=combinedDF['cluster_predicted'].value_counts().index,
                       hue=combinedDF['butter'],ax=axs[1])
         sns.countplot(x=combinedDF['cluster_predicted'],
                       order=combinedDF['cluster_predicted'].value_counts().index,
                       hue=combinedDF['waffles'],ax=axs[2])

         plt.tight_layout()
         plt.show()
```



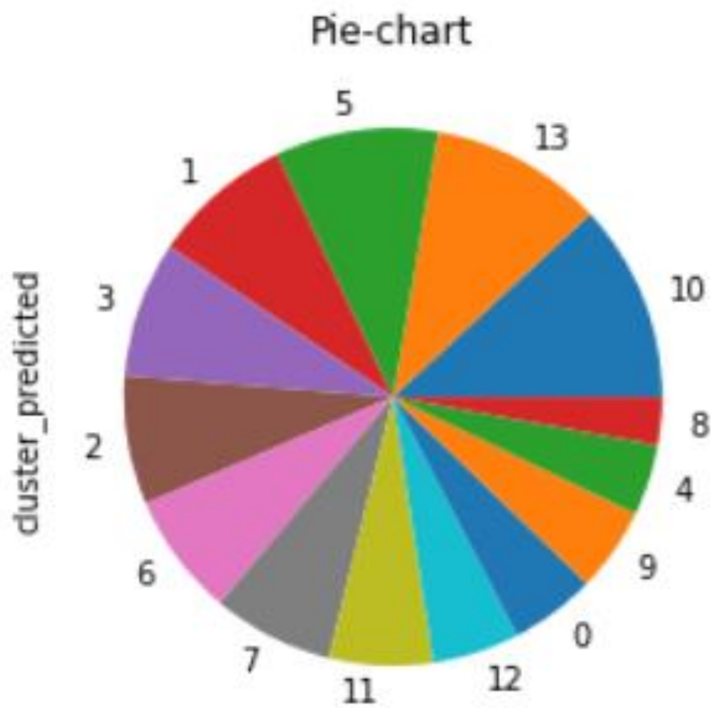
Plotting barplot for *cluster_predicted*:

```
In [84]: combinedDF['cluster_predicted'].value_counts().plot(kind='barh')

Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1c60babe518>
```



Plotting Pie Chart for *cluster_predicted*:

```
In [88]: combinedDF['cluster_predicted'].value_counts().plot(kind='pie', title = 'Pie-chart' )
```



VERIFICATION USING CLASSIFICATION

We have created clusters using KMeans clustering and given a label to our dataset rows representing the cluster that row belongs to. Now we need to check the accuracy of these formed clusters using classification algorithm.
Here we pick the **Logistic Regression** algorithm.

Logistic Regression:

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Preparation of Data for Logistic Regression:

Splitting dataset into features and label

```
In [89]: x = combinedDF.drop(['cluster_predicted'], axis = 1).values
         y = combinedDF['cluster_predicted'].values
```

```
In [90]: x

Out[90]: array([[1., 0., 1., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [1., 1., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 1., 0., 1.],
                [0., 1., 0., ..., 1., 0., 1.],
                [0., 0., 0., ..., 1., 1., 0.]])
```

```
In [93]: x.shape

Out[93]: (24885, 48)
```

In [91]:
```
y
```
Out[91]:
```
array([ 2, 13, 11, ...,  1,  4,  2])
```

In [94]:
```
y.shape
```
Out[94]:
```
(24885,)
```

Splitting dataset into train and test sets:

In [95]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

In [96]:
```
x_train.shape
```
Out[96]:
```
(19908, 48)
```

In [97]:
```
x_test.shape
```
Out[97]:
```
(4977, 48)
```

In [98]:
```
y_train.shape
```
Out[98]:
```
(19908,)
```

In [99]:
```
y_test.shape
```
Out[99]:
```
(4977,)
```

Scaling the data:

In [100]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train[:,:] = sc.fit_transform(x_train[:, :])
x_test[:,:] = sc.fit_transform(x_test[:,:])
```

```
In [101]: x_train
```

```
Out[101]: array([[ 2.65582342,  1.45296631,  2.6448404 , ..., -0.67717678,
                  -0.34777284, -0.52098632],
                 [-0.37653106,  1.45296631, -0.37809465, ...,  1.47671927,
                  -0.34777284,  1.91943619],
                 [-0.37653106, -0.6882472 , -0.37809465, ...,  1.47671927,
                  -0.34777284, -0.52098632],
                 ...,
                 [-0.37653106, -0.6882472 , -0.37809465, ..., -0.67717678,
                  -0.34777284,  1.91943619],
                 [-0.37653106,  1.45296631, -0.37809465, ...,  1.47671927,
                  -0.34777284,  1.91943619],
                 [-0.37653106, -0.6882472 , -0.37809465, ...,  1.47671927,
                  -0.34777284,  1.91943619]])
```

```
In [102]: x_test
```

```
Out[102]: array([[-0.39407167,  1.41229772, -0.38207596, ..., -0.68104591,
                  -0.35570752, -0.51927147],
                 [ 2.53760948,  1.41229772, -0.38207596, ..., -0.68104591,
                  -0.35570752,  1.92577498],
                 [-0.39407167, -0.708066  , -0.38207596, ..., -0.68104591,
                  -0.35570752, -0.51927147],
                 ...,
                 [-0.39407167, -0.708066  , -0.38207596, ..., -0.68104591,
                  -0.35570752, -0.51927147],
                 [-0.39407167, -0.708066  , -0.38207596, ..., -0.68104591,
                   2.81129844, -0.51927147],
                 [-0.39407167,  1.41229772,  2.6172806 , ..., -0.68104591,
                   2.81129844,  1.92577498]])
```

Importing *LogisticRegression* library from *sklearn*:

```
In [105]: from sklearn.linear_model import LogisticRegression
          log_reg = LogisticRegression()
```

Fitting the model to our data

```
In [106]: log_reg.fit(x_train, y_train)
          y_pred = log_reg.predict(x_test)
```

Checking Accuracy of our model using *accuracy_score*:

```
In [104]: from sklearn.metrics import accuracy_score
          a=accuracy_score(y_test,y_pred)
          a

Out[104]: 0.9628290134619248
```

We got accuracy of 96.28%.

CONCLUSION

Result of problem statement 1:

The user with <u>user_id = 269335</u> has bought 72 items in their lifetime.

Result of problem statement 2:

The *df* dataframe below contains the item name and the customer id of the customer who has bought that item the most.

```
In [111]: df = pd.DataFrame(item_fav_customer)
          df.columns = ['item', 'customer id']
          df
```

	item	customer id
0	dishwashing	956666
1	spaghetti sauce	1341188
2	poultry	334664
3	pork	1374100
4	beef	366155
5	laundry detergent	917199
6	shampoo	791038
7	tea	920002
8	frozen vegetables	1199670
9	coffee	996380
10	juice	255546
11	grapefruit	1433799
12	soap	1003550
13	sugar	1301034
14	soda	397623
15	lettuce	31625
16	dinner rolls	364868
17	pasta	289360
18	waffles	217277
19	bagels	820788
20	cheeses	884172
21	yogurt	943163
22	milk	837807
23	broccoli	297185
24	apples	1303742
25	cucumbers	80215
26	berries	384935
27	sandwich bags	360336
28	hand soap	394348
29	butter	478446
30	cauliflower	1198106
31	aluminum foil	143741
32	cereals	367872
33	cherries	109578
34	eggs	172120
35	ketchup	133355
36	canned vegetables	238495
37	carrots	743501
38	bananas	1218645
39	pet items	1433188
40	shaving cream	31625
41	sandwich loaves	599172
42	flour	1076958
43	tortillas	1485538
44	toilet paper	1425746
45	paper towels	1077463
46	ice cream	269335
47	baby items	73071

Result of problem statement 3:

There exist 14 clusters for the given grocery items that are likely to be bought together. They are:

Cluster 1	Apples, Bananas, Berries, Cherries, Grapefruit
Cluster 2	Broccoli, Carrots, Cauliflower, Cucumber, Lettuce
Cluster 3	Bagels, Dinner Rolls, Sandwich Loaves, Tortillas
Cluster 4	Coffee, Juice, Soda, Tea
Cluster 5	Butter, Cheeses, Eggs, Milk, Yoghurt
Cluster 6	Beef, Pork, Poultry
Cluster 7	Canned Vegetables, Ketchup, Spaghetti Sauce
Cluster 8	Hand Soap, Shampoo, Shaving Cream, Soap
Cluster 9	Aluminum Foil, Paper Towels, Sandwich Bags, Toilet Paper
Cluster 10	Cereals, Flour, Pasta, Waffles
Cluster 11	Dishwashing, Laundry Detergent
Cluster 12	Frozen Vegetables, Ice Cream
Cluster 13	Pet items, Sugar
Cluster 14	Baby items

The clusters were formed by plotting a Dendrogram using Hierarchical Clustering. The clusters were fit to the data using KMeans Clustering.

When the accuracy of the formed clusters was tested using Logistic Regression Classification, we got an accuracy of 96.28%.

REFERENCES

Links:

https://scikit-learn.org/stable/modules/clustering.html

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3477851/

https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html#:~:text=Plot%20the%20hierarchical%20clustering%20as,singleton%20cluster%20and%20its%20children.&text=It%20is%20also%20the%20cophenetic,in%20the%20two%20children%20clusters.

https://matplotlib.org/3.3.0/api/_as_gen/matplotlib.pyplot.scatter.html

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.barh.html

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html