Instacart Market Basket Analysis

MITA Capstone Project

Surabhi Kator

Table of Contents

1. Introduction

Nowadays**,** one of the challenges for supermarket chains is to offer recommendation services to their customers. Detecting the customers' purchase habits and analyzing the association between different products is a crucial challenge for effective marketing policies and engagement strategies. In fact, customers' patterns keep altering and can experience changes due to the seasonality of the products or retail policies or changes in ones' diet and personal preferences.

A retail company selling commodity goods in its outlets thought of mining and going through the data of the customers and discovered a peculiar connection between purchasing stock that people who bought beers also bought diapers, and most of the time people who bought diapers also bought beer with it. After an in-depth study of this trend, it was found that about 95% of those who would buy diapers on weekends used to buy beers with it. With this, the business decided to put both products in the same aisle together and increase their price. Thus, a satisfactory solution to solve such a challenge is to analyze the ordering habits of the customers to know the association between different products, and then suggesting these products to the user the next time he adds something to the cart.

Market basket analysis is one such technique used by many retails to understand the purchasing patterns of the customers. It is based upon the theory that if you buy a certain group of items, you are more or less likely to buy another group of items from the store. For example, if you are in a Supermarket and you buy eggs, then you are more likely to buy bread and Milk than somebody who did not buy eggs.

In this project, we are going to analyze the groceries dataset provided by Instacart to identify the ordering patterns of the customers, to identify which items are bought together and which items have more demand. Then, we will see the association between different items at Instacart, so that we will get to know which items are usually added together to the cart. After that, we will build a recommender system, that will predict the items to be bought together.

2. Dataset

In this section, we are going to discuss in detail the dataset provided by Instacart. The dataset for this project is a relational set of files describing all the customer's orders over time. The data used for this project is from the 2017 competition that Instacart hosted on

Kaggle.com. The data contains a sample of over 3 million grocery orders from more than 200,000 Instacart users.

A total of six datasets were imported. The following section will explore each data sets in further detail.

1. Orders

   This dataset has 3.4 million rows and 206K users. The attributes of this dataset are as follows:

Attribute	Description
Order_id	This is a unique order identifier
User_id	This is a unique customer identifier
Eval_set	Which evaluation set the particular order belongs in
Order_number	The order sequence number for this user(1=first,n=nth)
Order_dow	The day of the week the order was placed on
Order_hour_of_day	The hour of the day the order was placed on
Days_since_prior	Days since the last order

2. Products

   This dataset has 50K rows and describes the product names, aisles, and departments associated with the products.

Attribute	Description
Product_id	This is a unique product identifier
Product_name	This is the name of the product
Aisle_id	This gives the aisles associated with the product (foreign key)
Department_id	This gives the department associated with the product (foreign key)

3. Aisles

   This dataset has 134 rows and describes the aisles of Instacart for the products.

Attribute	Description
Aisle_id	This is a unique aisle identifier
Aisle	The name of each aisle

4. Departments

   This dataset has 21 rows and describes the departments of Instacart for the products.

Attribute	Description
Department_id	Department identifier
Department	The name of the department

5. Orders_products SET

   This dataset contains 30m+ records and is divided into two datasets—the orders_train_prior set and the order test prior set. The attributes for both datasets are the same.

Attribute	Description
Order_id	This is the unique order id
Product_id	Unique product id to identify each product
Add_to_cart_order	Order in which each product was added to cart
Reordered	1 if this product has been ordered by this ser in the past,0 otherwise

The SET is one of the four following evaluation sets (eval_set in orders):

- "prior": orders prior to that user's most recent order (~3.2m orders)
- "train": training data supplied to participants (~131k orders)
- "test": test data reserved for machine learning competitions (~75k orders)

3. Problem Statement

From our Instacart dataset, we have some very interesting variables that we can use to understand the ordering patterns of the users. We need to answer the following questions for our data:

- What are the distributions of orders by day of the week, the hour of the day?

- Reorder Interval - How often do customers put in a new order? Distribution of the reorder interval?
- What are the top products, aisles, departments ordered?
- What are the types of products reordered? Are there differences between products reordered within a week compared to within a month?
- What is the association between the frequently bought items?
- Can we recommend the related items based on the existing order history?

The main goal of our project would be to use our dataset to uncover the ordering patterns, show the association between the items, and recommend the items to the users.

4. Methodology

The project's main objective is to analyze the data first, and then uncover the associations between different products based on association rule mining, and then recommending the items to the user.

This project is implemented using Jupyter Notebook Python.

It is divided into three main modules:

The first module is about data analysis. Exploratory Data Analysis is performed to uncover hidden trends and patterns in the dataset.

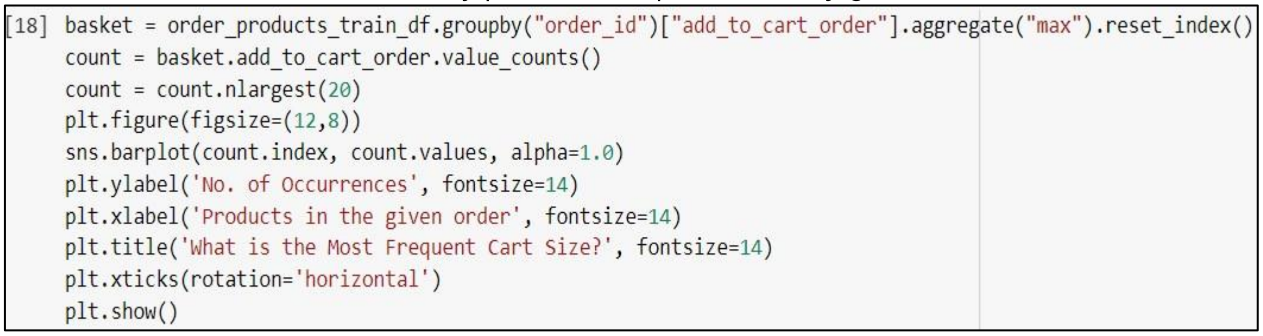
Then, association rule mining using the Apriori algorithm is implemented so that we get to know the association between different products in our dataset. This mining is implemented using the If then rule, that is, if the user adds x to his cart, then how likely or less likely does he add y to the cart.

And then, the Recommendation system is implemented to actually recommend products to the user based on the previous transactions. For this project, we will be implementing the collaborative filtering recommendation system using the LigthFm library.

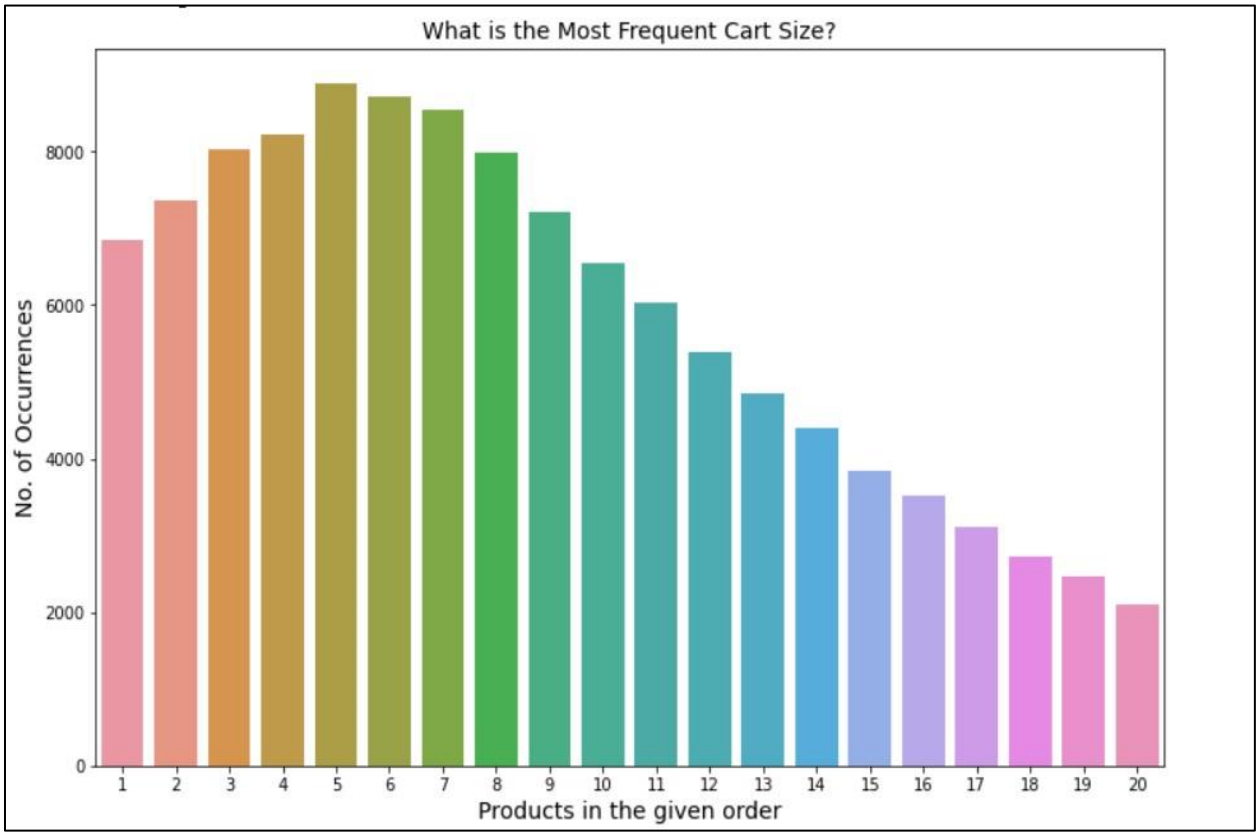
5. Exploratory Data Analysis and Visualization

In statistics, exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. EDA will help us uncover the trends and patterns in our data by using different attributes of the data. So, let us look at the purchasing patterns of the users at Instacart.

First, let us have a look at how many products are present in any given order.

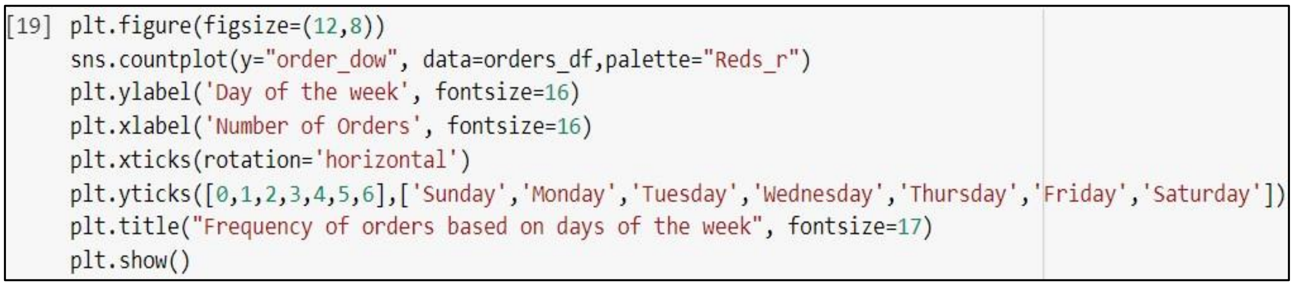


```
[18] basket = order_products_train_df.groupby("order_id")["add_to_cart_order"].aggregate("max").reset_index()
     count = basket.add_to_cart_order.value_counts()
     count = count.nlargest(20)
     plt.figure(figsize=(12,8))
     sns.barplot(count.index, count.values, alpha=1.0)
     plt.ylabel('No. of Occurrences', fontsize=14)
     plt.xlabel('Products in the given order', fontsize=14)
     plt.title('What is the Most Frequent Cart Size?', fontsize=14)
     plt.xticks(rotation='horizontal')
     plt.show()
```

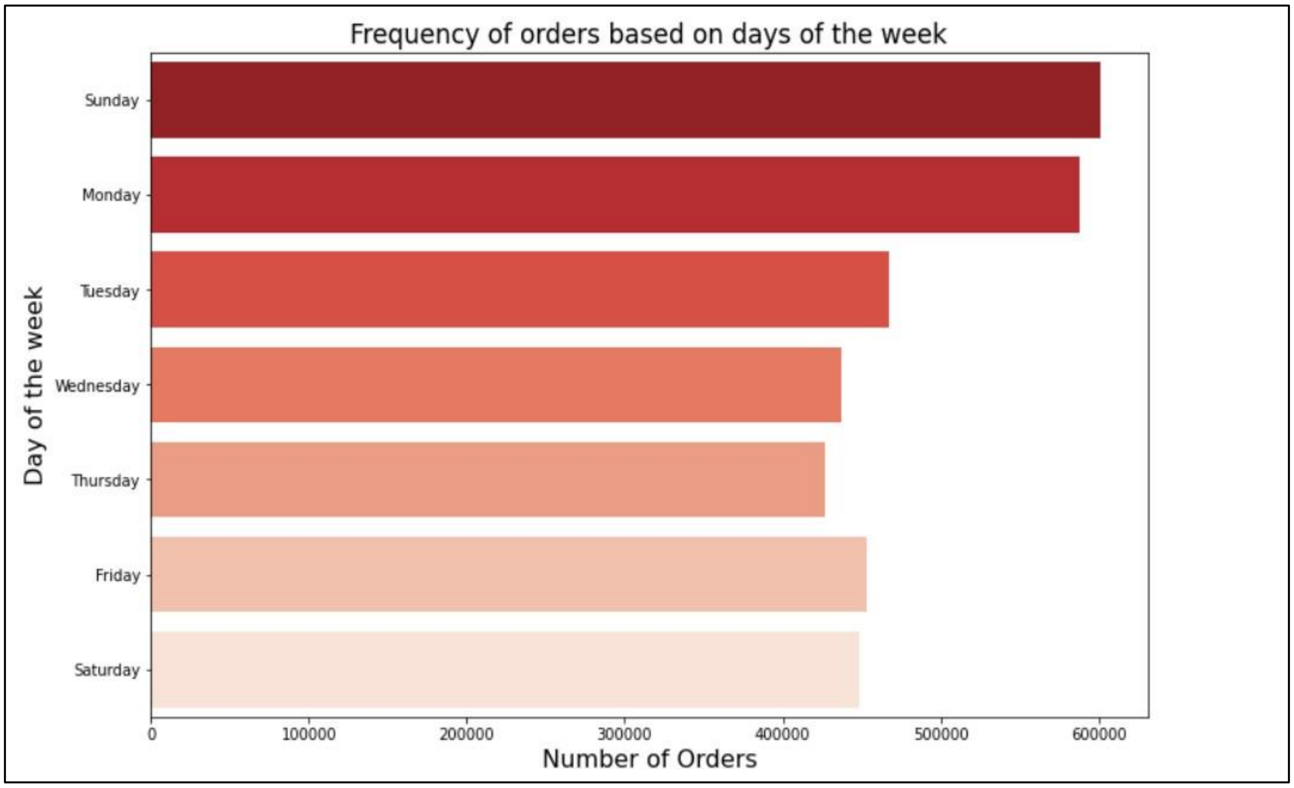


From the above plot, we can see that mostly five items or products are ordered per order. Also, we can see peaks at 6 and 7. So on average, 5-7 products are the most frequent cart size.

Now, let us have a look at the frequency of orders based on the days of the week. We will use the "order_dow" attribute from the "Orders.csv" dataset to get the output. It ranges from 0 to 6. Here, 0 we must assume is Sunday, followed by Monday for one, and so on.

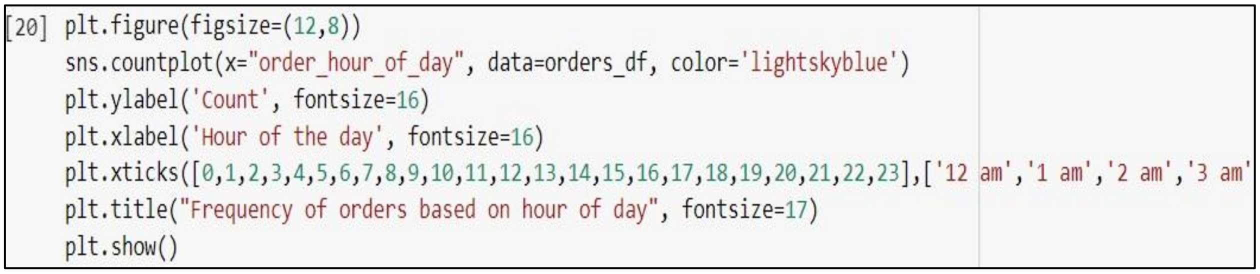


```
[19] plt.figure(figsize=(12,8))
     sns.countplot(y="order_dow", data=orders_df,palette="Reds_r")
     plt.ylabel('Day of the week', fontsize=16)
     plt.xlabel('Number of Orders', fontsize=16)
     plt.xticks(rotation='horizontal')
     plt.yticks([0,1,2,3,4,5,6],['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday'])
     plt.title("Frequency of orders based on days of the week", fontsize=17)
     plt.show()
```

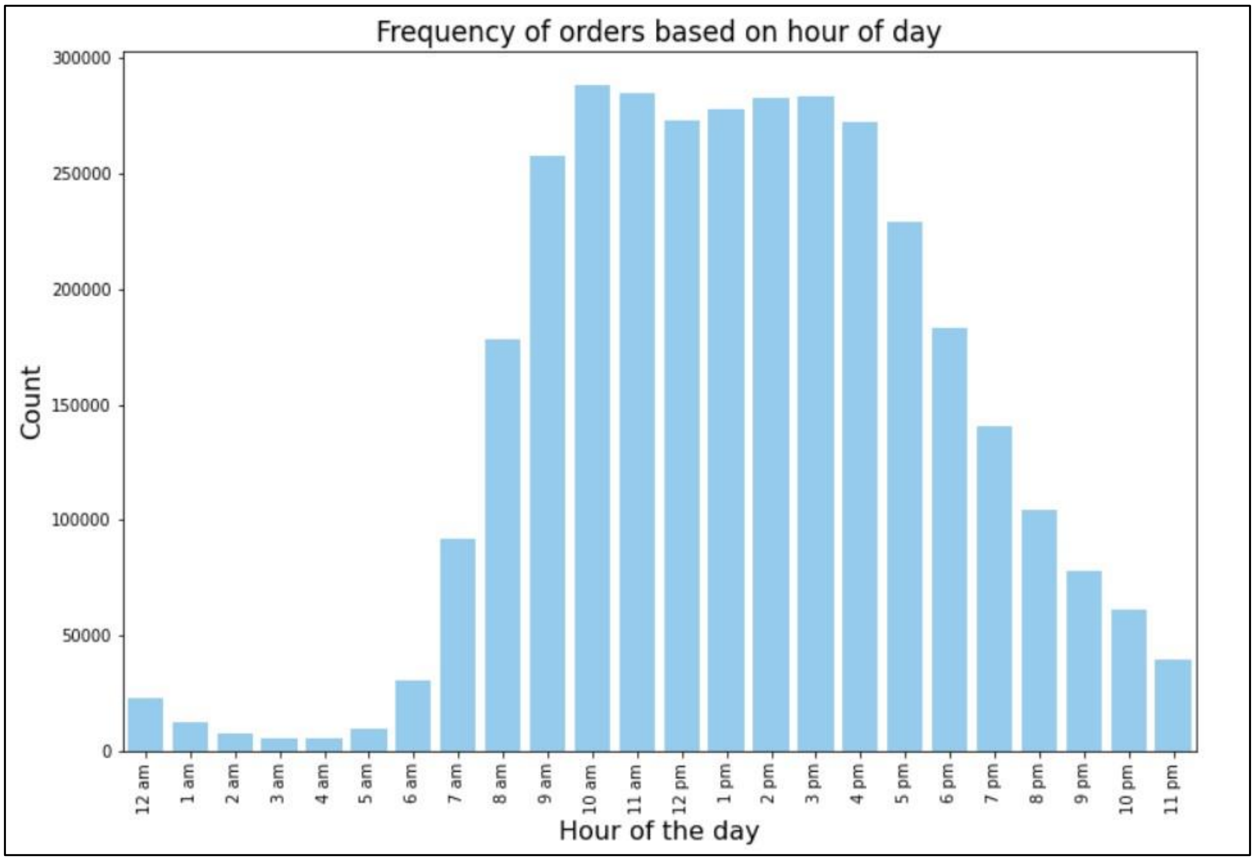


From this graph, we can see that the number of orders is more on Sundays and Mondays, so it seems that people shop more either at the end of the week or at the start of the week. Also, it seems that people shop the least in the middle of the week, that is, on Wednesdays and Thursdays since we can see fewer spikes in the middle of the week.

Now, let us look at the ordering pattern of the users based on the hour of the day. For this, we will use the variable "order_hour_of_day" from the "Orders.csv" dataset. The values range from 0 to 23. Here, we must assume that 0 is 12 am at night, 1 for 1 am, and so on.

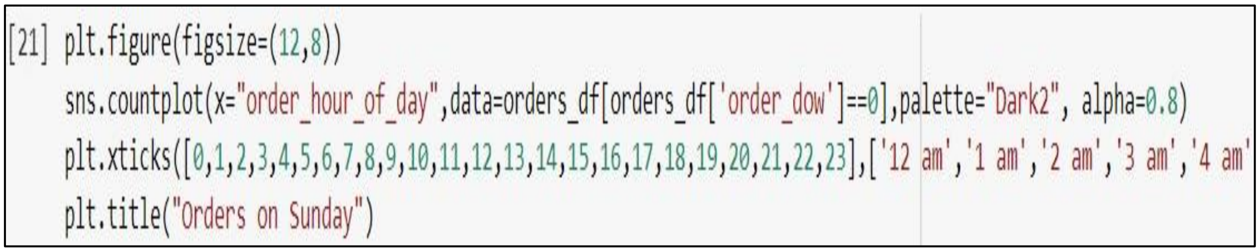


```
[20] plt.figure(figsize=(12,8))
     sns.countplot(x="order_hour_of_day", data=orders_df, color='lightskyblue')
     plt.ylabel('Count', fontsize=16)
     plt.xlabel('Hour of the day', fontsize=16)
     plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23],['12 am','1 am','2 am','3 am'
     plt.title("Frequency of orders based on hour of day", fontsize=17)
     plt.show()
```

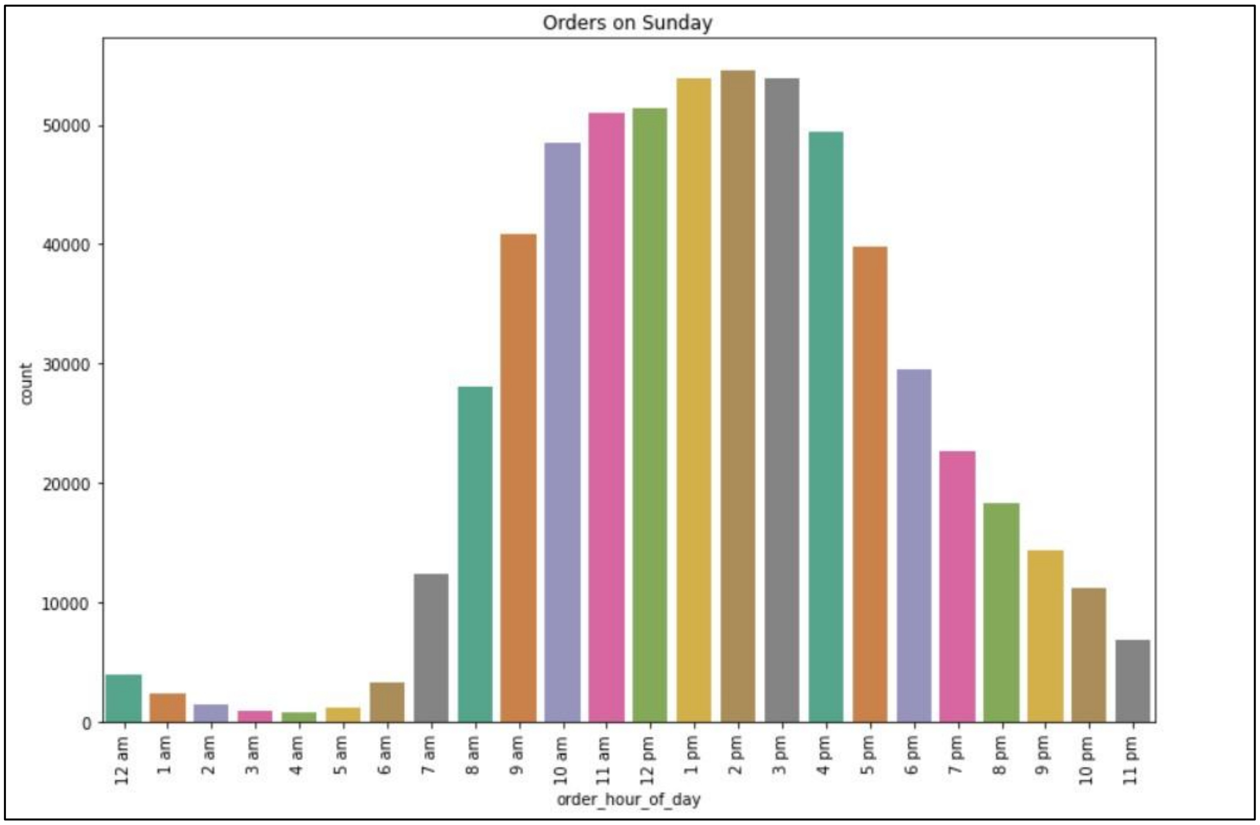


From the plot, we can clearly see that the maximum number of orders were placed at 10 am and 11 am, which makes sense because that is the time when people usually start their day. Also, the least number of orders were placed at 3 am and 4 am because most people are asleep during that period of the day. Then, we can also see spikes at 2 pm and 3 pm.

Now, to know more about the ordering patterns, let us have a look at when the orders are placed on the day of the week and hour of the day. Let us have a look at the ordering pattern on a Sunday.

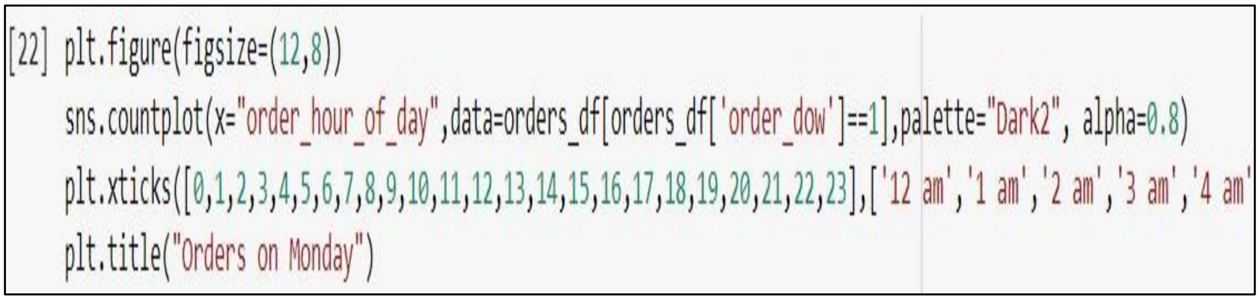


```
[21] plt.figure(figsize=(12,8))
     sns.countplot(x="order_hour_of_day",data=orders_df[orders_df['order_dow']==0],palette="Dark2", alpha=0.8)
     plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23],['12 am','1 am','2 am','3 am','4 am'
     plt.title("Orders on Sunday")
```

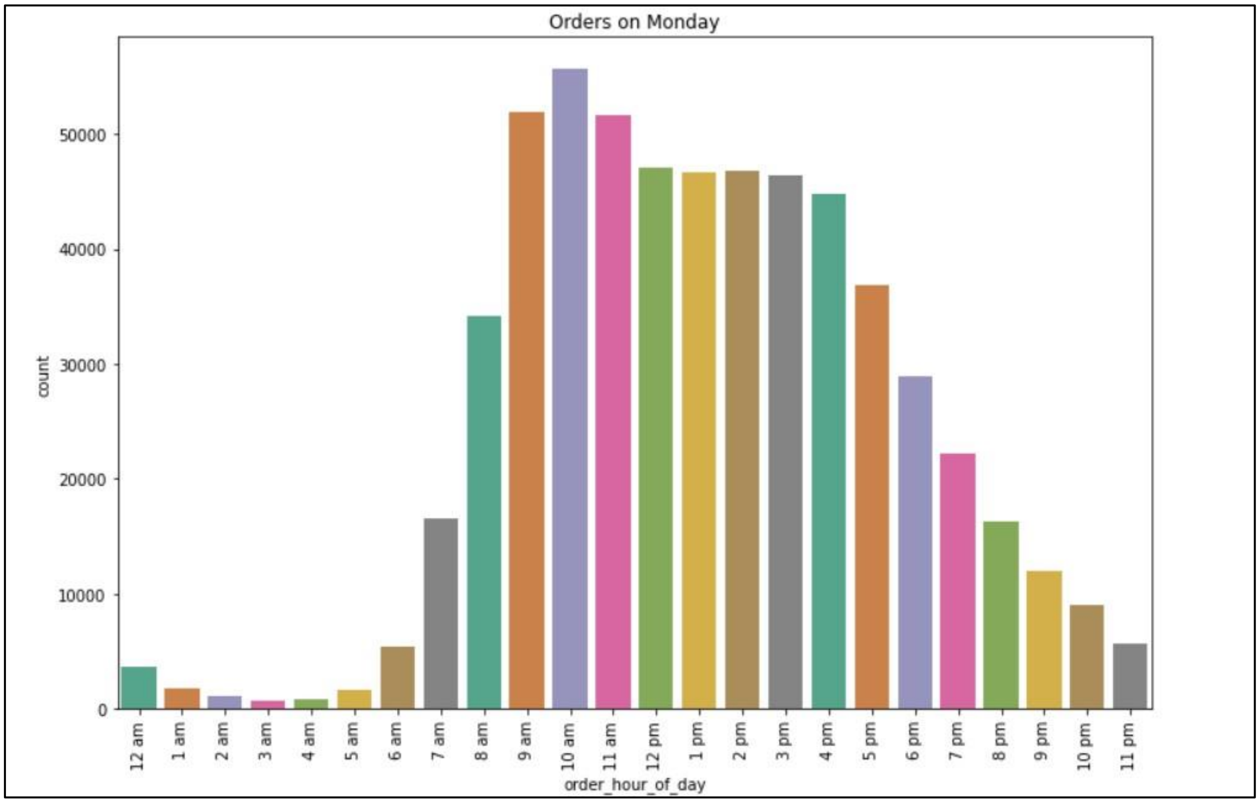


This plot shows the time of orders placed on Sunday. We can see that most of the orders were placed at 2 pm and 3 pm on a Sunday.

Let us have a look at the ordering pattern on Monday.



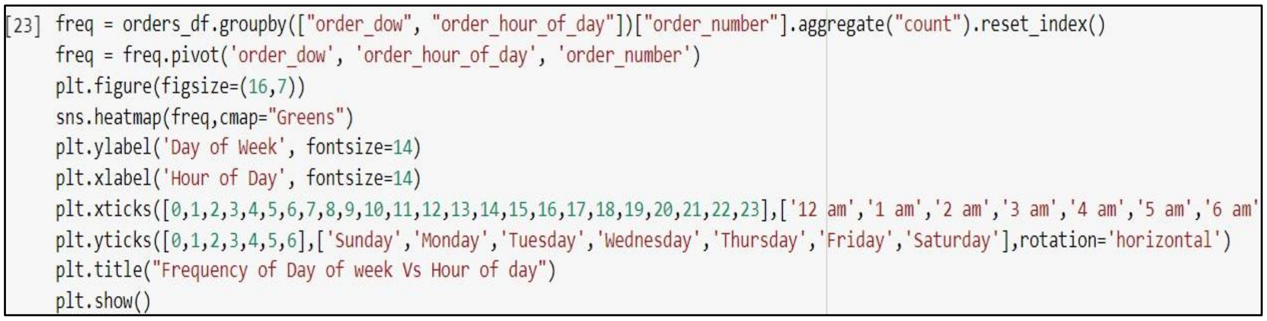
```
[22] plt.figure(figsize=(12,8))
     sns.countplot(x="order_hour_of_day",data=orders_df[orders_df['order_dow']==1],palette="Dark2", alpha=0.8)
     plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23],['12 am','1 am','2 am','3 am','4 am'
     plt.title("Orders on Monday")
```



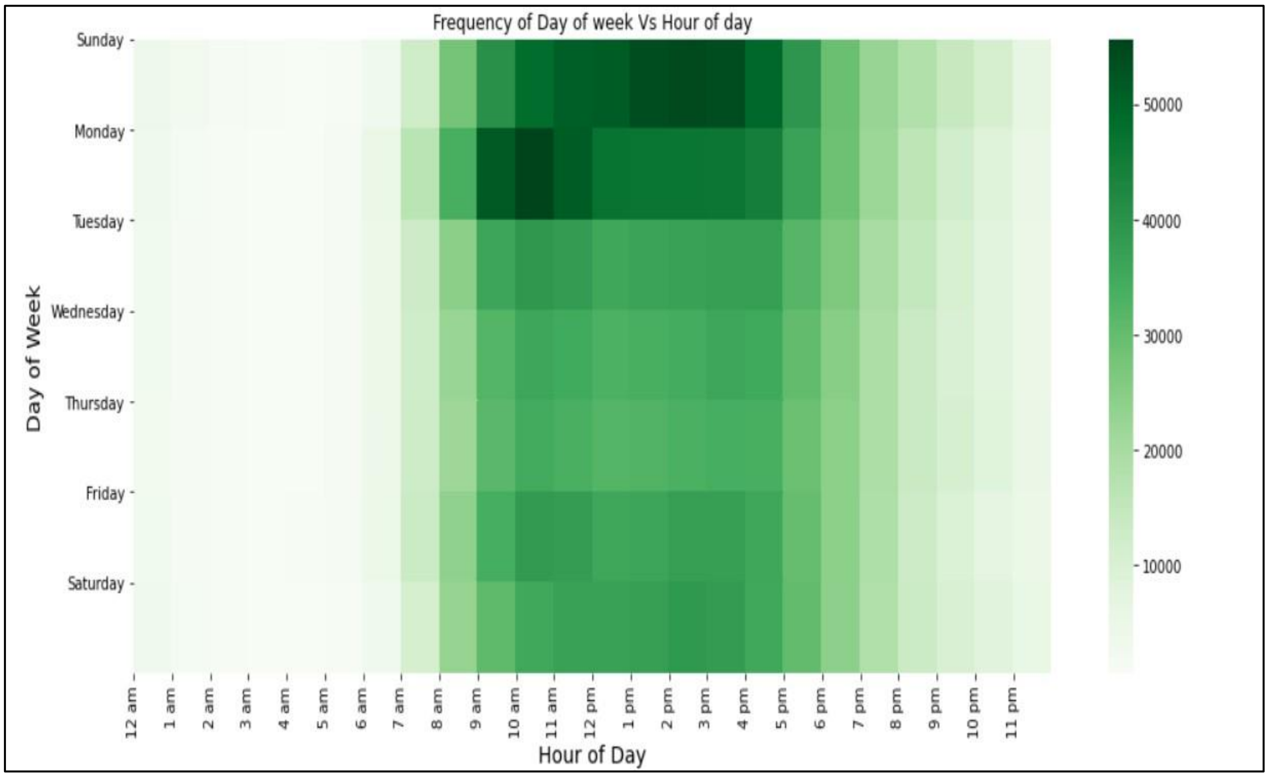
On Monday, we can see most of the orders were placed at 10 am. So, most of the orders were placed from 9 am to 11 am.

Now, let us look at the correlation between the day of the week and the hour of the day. For this, we will use tow variables "order_dow" and "order_hour_of_day" from "Orders.csv" dataset.

We will use a heatmap to show the correlation between these two variables.

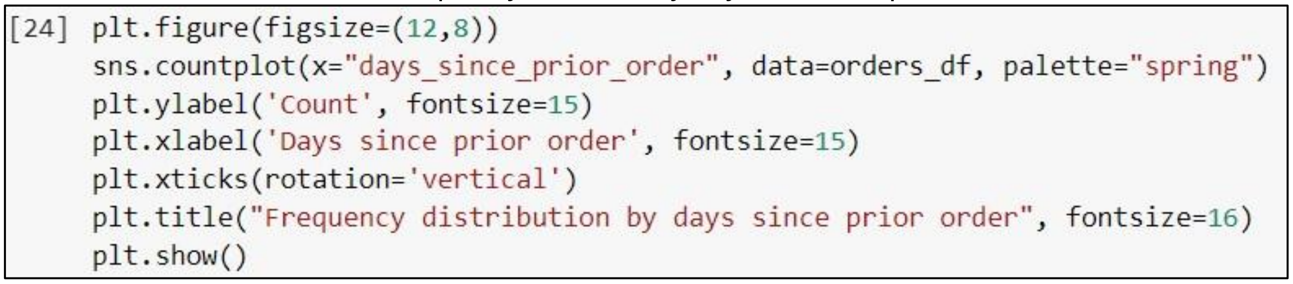


```
[23] freq = orders_df.groupby(["order_dow", "order_hour_of_day"])["order_number"].aggregate("count").reset_index()
     freq = freq.pivot('order_dow', 'order_hour_of_day', 'order_number')
     plt.figure(figsize=(16,7))
     sns.heatmap(freq,cmap="Greens")
     plt.ylabel('Day of Week', fontsize=14)
     plt.xlabel('Hour of Day', fontsize=14)
     plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23],['12 am','1 am','2 am','3 am','4 am','5 am','6 am'
     plt.yticks([0,1,2,3,4,5,6],['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday'],rotation='horizontal')
     plt.title("Frequency of Day of week Vs Hour of day")
     plt.show()
```

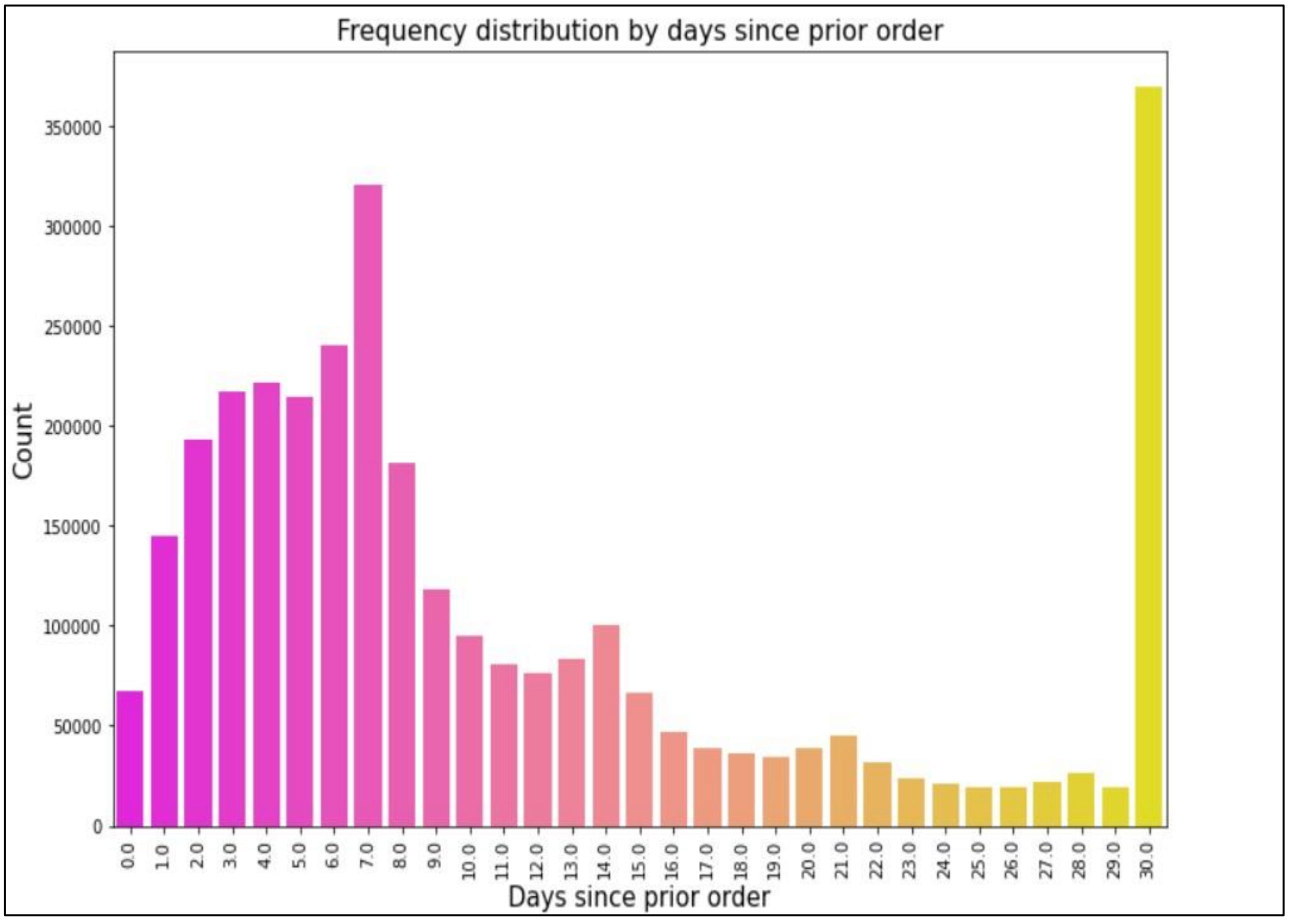


From this heatmap, we can see that the peak orders are in the afternoon on Sunday and on Monday between 9 am to 4 pm. So it looks like Sundays and Mondays are the prime days for ordering.

Now, let us have a look at the frequency of orders by days since the prior order.

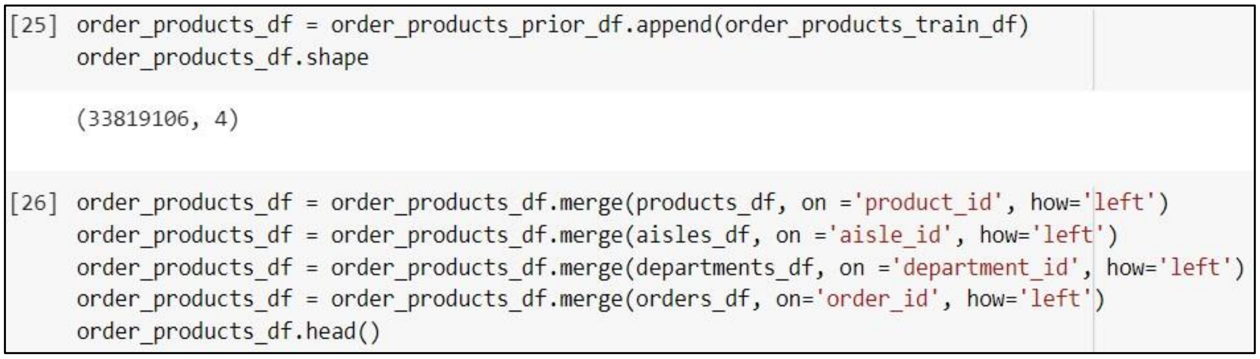


```
[24] plt.figure(figsize=(12,8))
     sns.countplot(x="days_since_prior_order", data=orders_df, palette="spring")
     plt.ylabel('Count', fontsize=15)
     plt.xlabel('Days since prior order', fontsize=15)
     plt.xticks(rotation='vertical')
     plt.title("Frequency distribution by days since prior order", fontsize=16)
     plt.show()
```



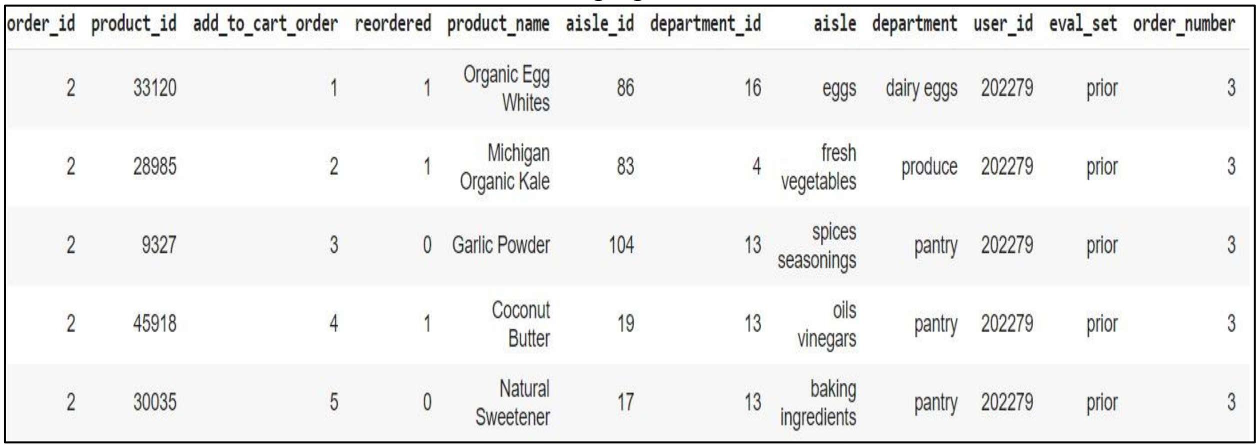
We can clearly see the products have been reordered every seven days and every 30 days. That means people usually place the next order after a week or then after a month. Also, there are small spikes on days 6, 14, and 28.

Now, let us combine the order_products_prior_df and order_products_train_df datasets into one dataframe order_products_df. Then, based on product_id, aisle_id, department_id, and order_id, we will merge the data from "Products.csv, " Aisles.csv, " Departments.csv," and "Orders.csv" into one data frame so that we will get all the important attributes from all these datasets into a single dataframe.



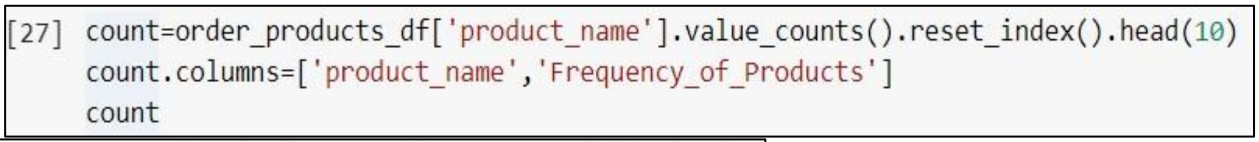
```
[25] order_products_df = order_products_prior_df.append(order_products_train_df)
     order_products_df.shape

     (33819106, 4)

[26] order_products_df = order_products_df.merge(products_df, on ='product_id', how='left')
     order_products_df = order_products_df.merge(aisles_df, on ='aisle_id', how='left')
     order_products_df = order_products_df.merge(departments_df, on ='department_id', how='left')
     order_products_df = order_products_df.merge(orders_df, on='order_id', how='left')
     order_products_df.head()
```

This is how the dataframe looks like after merging all the attributes.

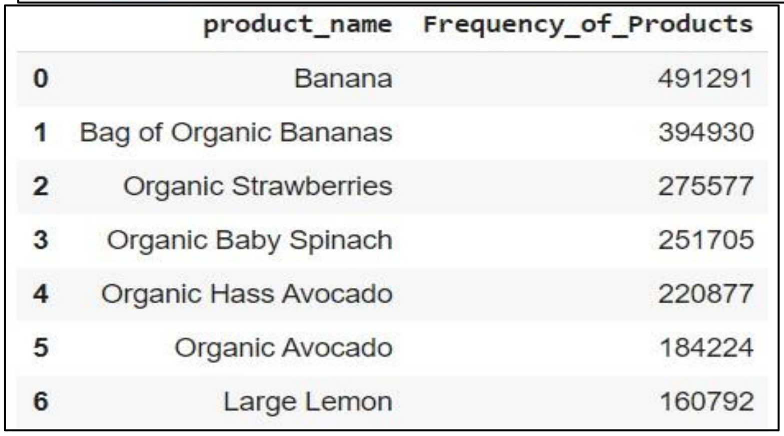


order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	aisle	department	user_id	eval_set	order_number
2	33120	1	1	Organic Egg Whites	86	16	eggs	dairy eggs	202279	prior	3
2	28985	2	1	Michigan Organic Kale	83	4	fresh vegetables	produce	202279	prior	3
2	9327	3	0	Garlic Powder	104	13	spices seasonings	pantry	202279	prior	3
2	45918	4	1	Coconut Butter	19	13	oils vinegars	pantry	202279	prior	3
2	30035	5	0	Natural Sweetener	17	13	baking ingredients	pantry	202279	prior	3

Now, after merging the datasets, we will count the occurrence of all the products in the orders.

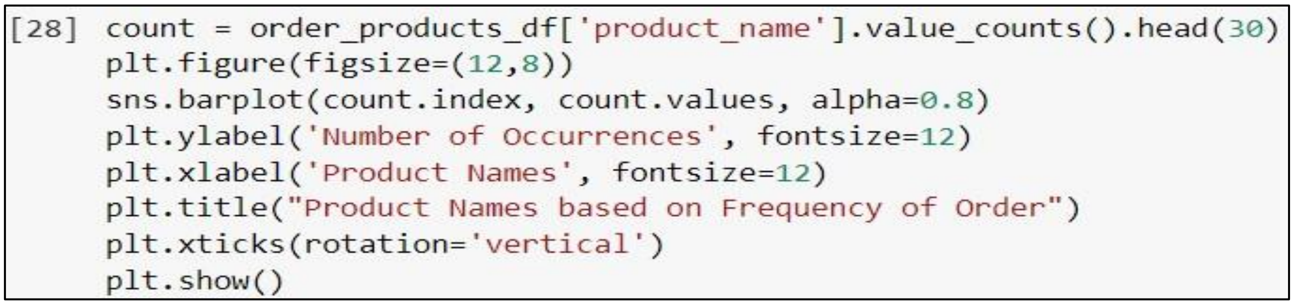


```
[27] count=order_products_df['product_name'].value_counts().reset_index().head(10)
     count.columns=['product_name','Frequency_of_Products']
     count
```

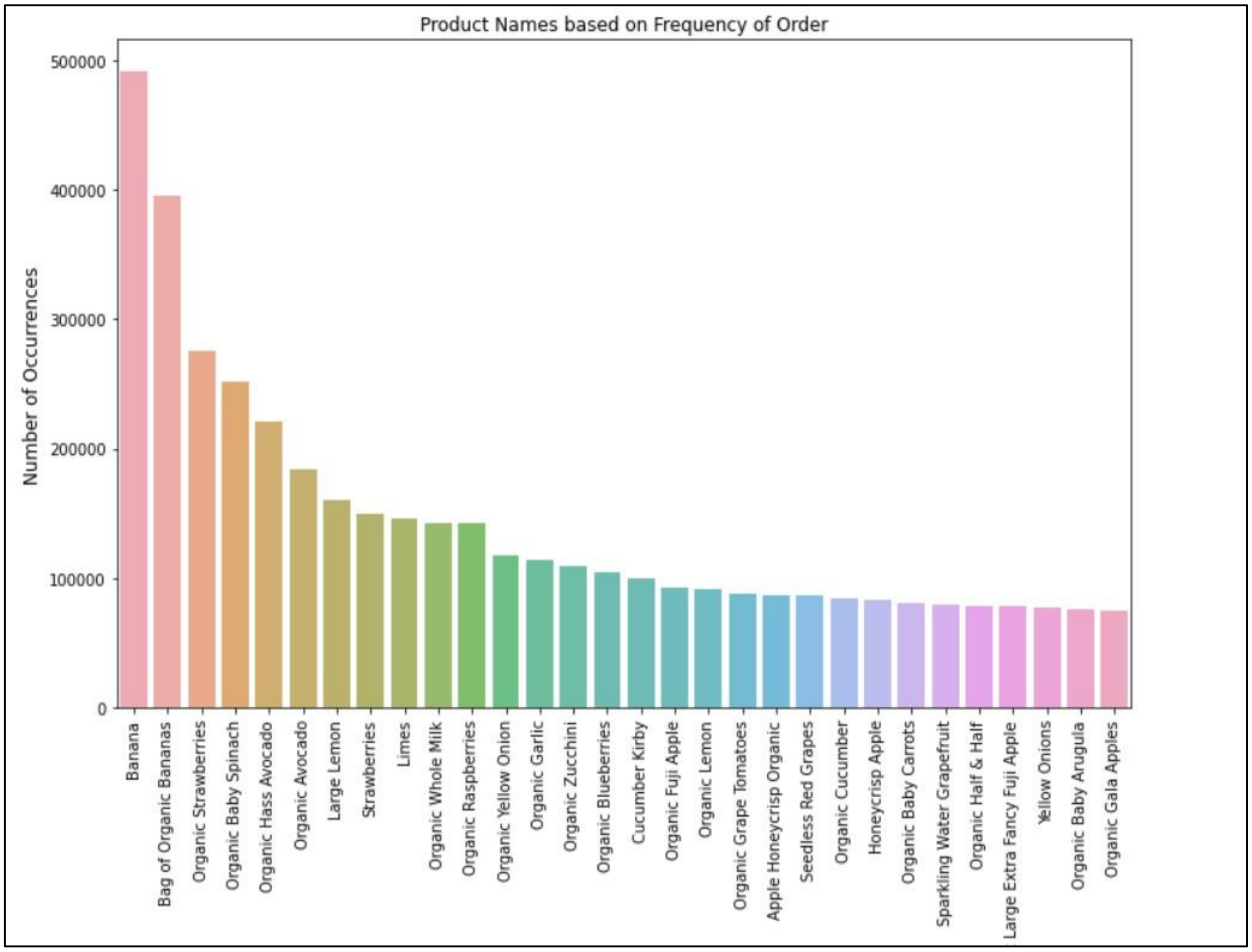


	product_name	Frequency_of_Products
0	Banana	491291
1	Bag of Organic Bananas	394930
2	Organic Strawberries	275577
3	Organic Baby Spinach	251705
4	Organic Hass Avocado	220877
5	Organic Avocado	184224
6	Large Lemon	160792

Now, let us plot the frequency of the products.

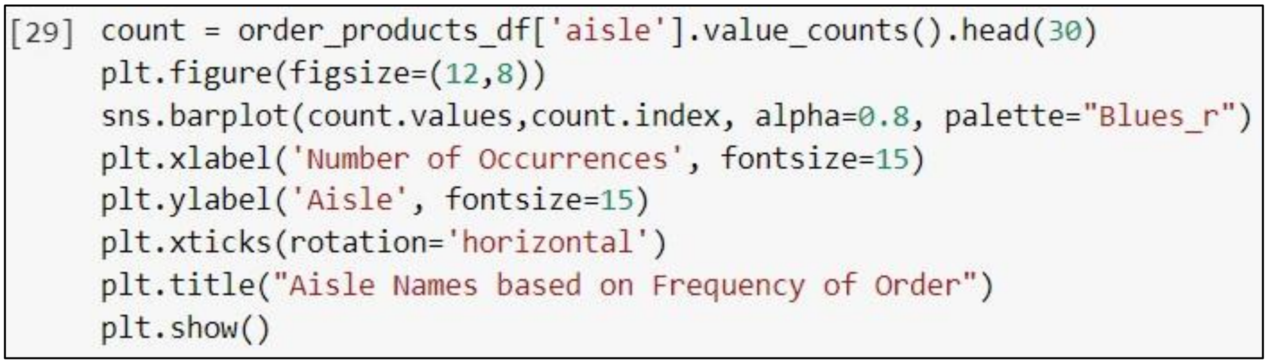


```
[28] count = order_products_df['product_name'].value_counts().head(30)
     plt.figure(figsize=(12,8))
     sns.barplot(count.index, count.values, alpha=0.8)
     plt.ylabel('Number of Occurrences', fontsize=12)
     plt.xlabel('Product Names', fontsize=12)
     plt.title("Product Names based on Frequency of Order")
     plt.xticks(rotation='vertical')
     plt.show()
```

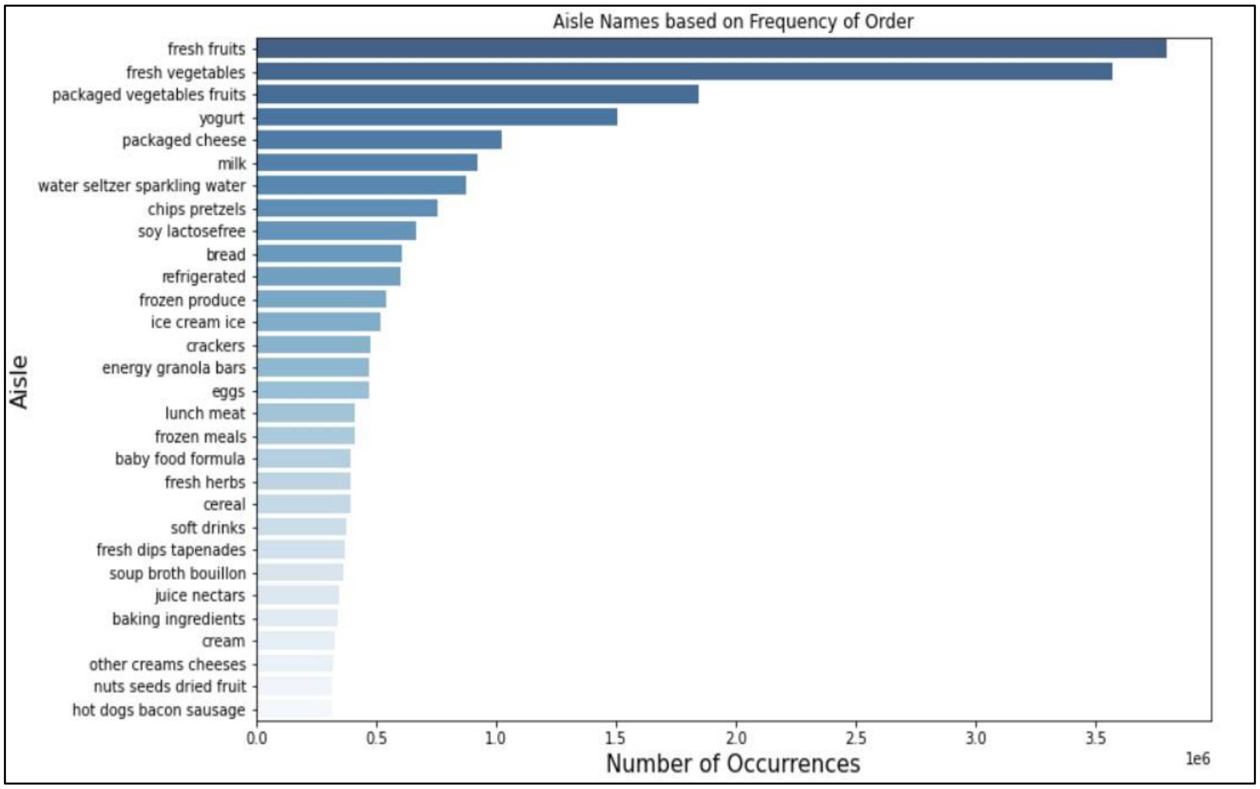


From the above visualization, we can see that Bananas and Organic Bananas are the most frequently ordered products at Instacart, followed by Organic Strawberries, Organic Baby Spinach, Organic Hass Avocado, Large Limes, Strawberries, and so on. So we can say that fruits and vegetables followed by Milk are the most frequently ordered products at Instacart.

Let us look at the frequency of orders across different aisles at Instacart.

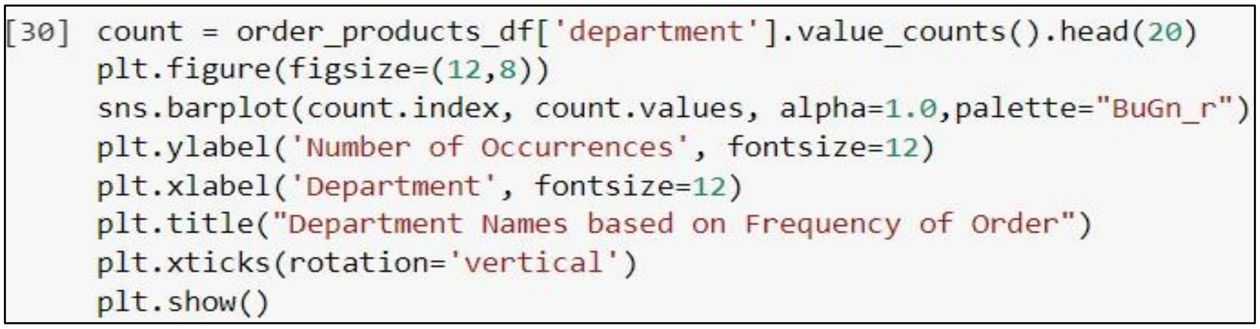


```
[29] count = order_products_df['aisle'].value_counts().head(30)
     plt.figure(figsize=(12,8))
     sns.barplot(count.values,count.index, alpha=0.8, palette="Blues_r")
     plt.xlabel('Number of Occurrences', fontsize=15)
     plt.ylabel('Aisle', fontsize=15)
     plt.xticks(rotation='horizontal')
     plt.title("Aisle Names based on Frequency of Order")
     plt.show()
```

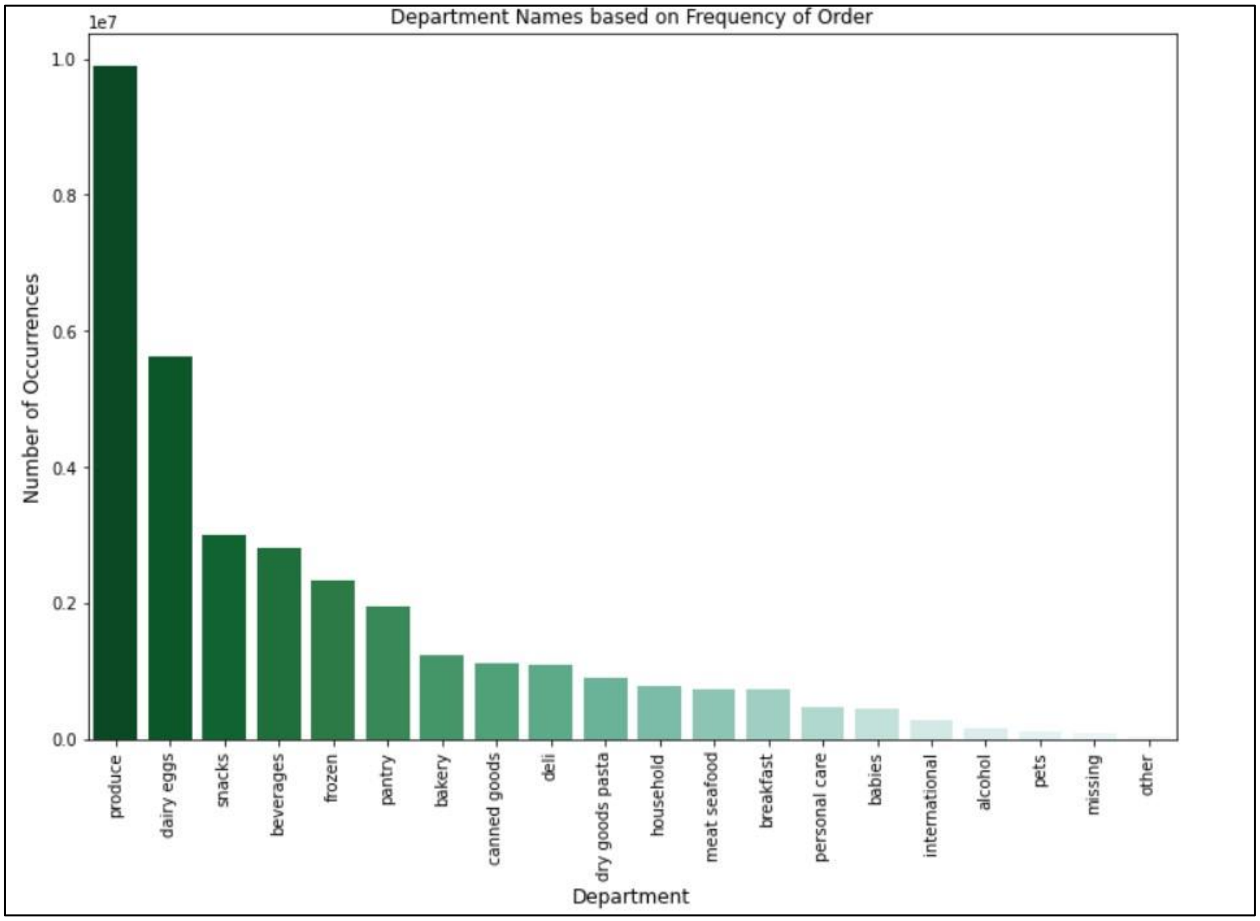


From the above visualization, we can see that fresh fruits and fresh vegetables are the most popular aisles at Instacart. Also, we can see demand for packaged vegetables, yogurt, packaged cheese, and Milk.

Now, let us have a look at the frequency of orders across different departments at Instacart.

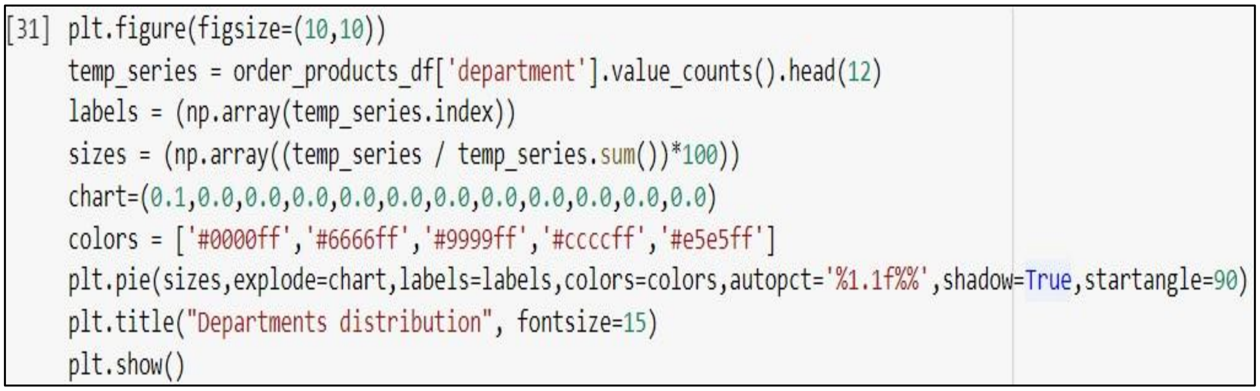


```
[30] count = order_products_df['department'].value_counts().head(20)
     plt.figure(figsize=(12,8))
     sns.barplot(count.index, count.values, alpha=1.0,palette="BuGn_r")
     plt.ylabel('Number of Occurrences', fontsize=12)
     plt.xlabel('Department', fontsize=12)
     plt.title("Department Names based on Frequency of Order")
     plt.xticks(rotation='vertical')
     plt.show()
```



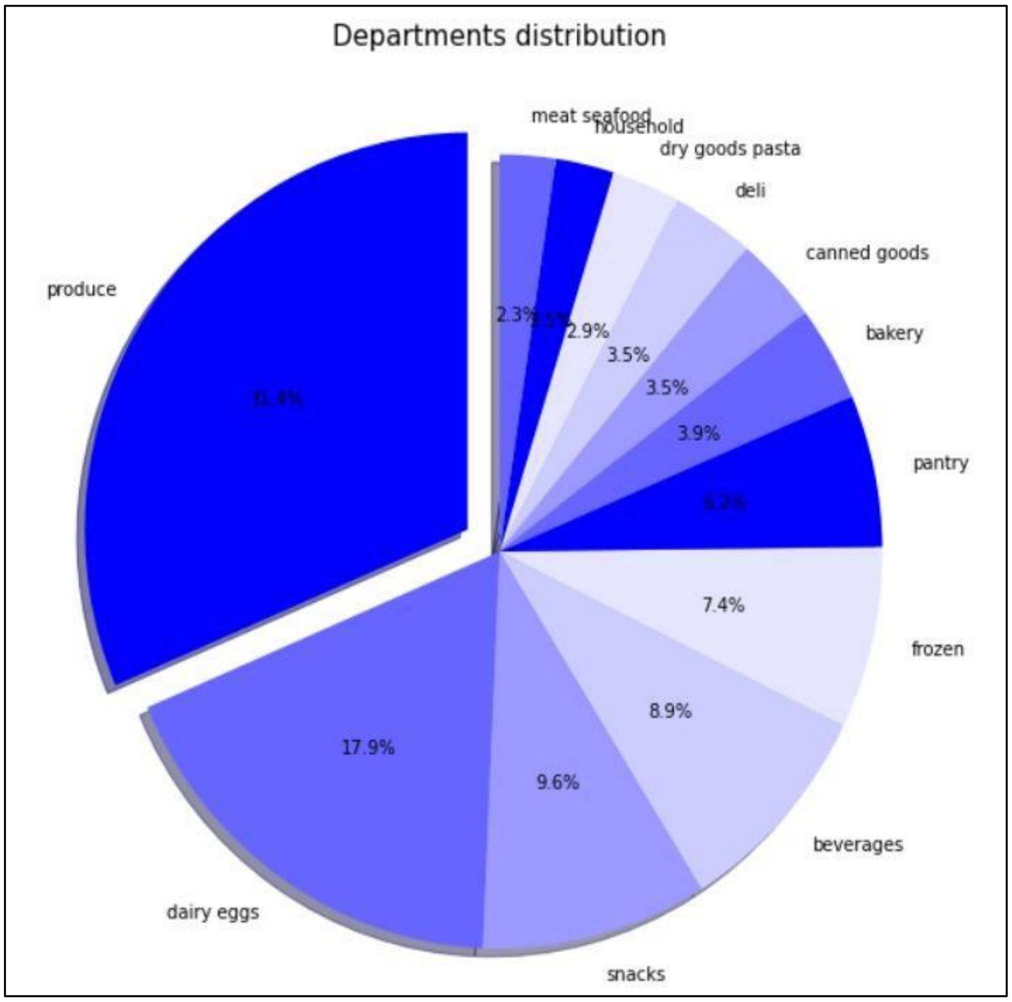
From this graph, we can see that the produce, dairy eggs, snacks and beverages are the most popular departments at Instacart.

Now, let us take a look at the department wise distribution of the products.

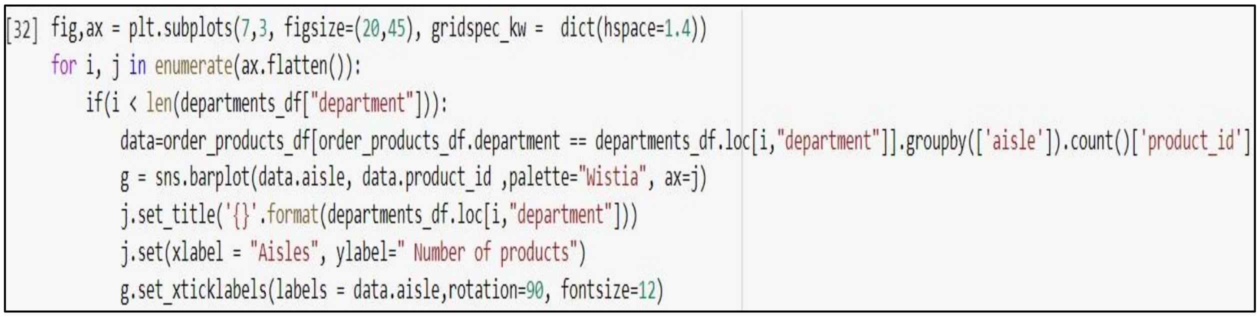


```
[31] plt.figure(figsize=(10,10))
     temp_series = order_products_df['department'].value_counts().head(12)
     labels = (np.array(temp_series.index))
     sizes = (np.array((temp_series / temp_series.sum())*100))
     chart=(0.1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)
     colors = ['#0000ff','#6666ff','#9999ff','#ccccff','#e5e5ff']
     plt.pie(sizes,explode=chart,labels=labels,colors=colors,autopct='%1.1f%%',shadow=True,startangle=90)
     plt.title("Departments distribution", fontsize=15)
     plt.show()
```

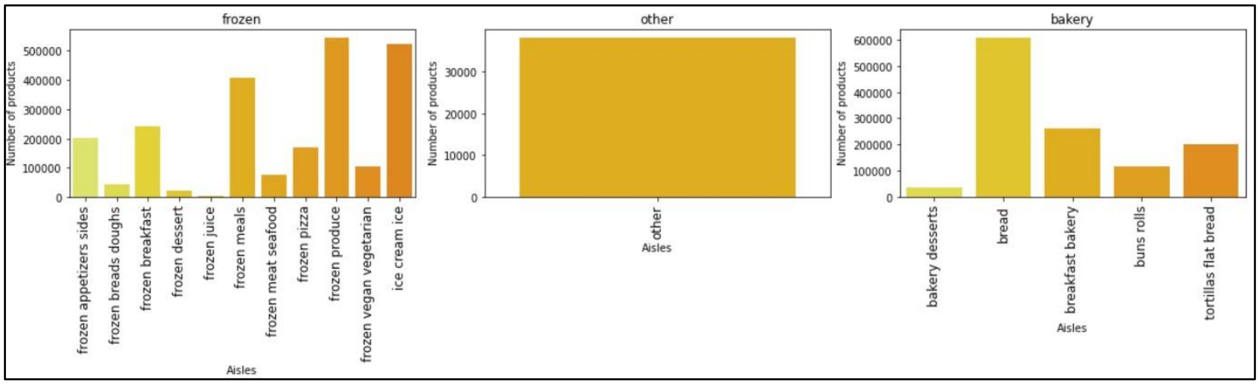
From the below pie chart we can see that the produce department has the most demand, which is 31.4% followed by dairy eggs which are 17.9% and then snacks which is 9.6%.

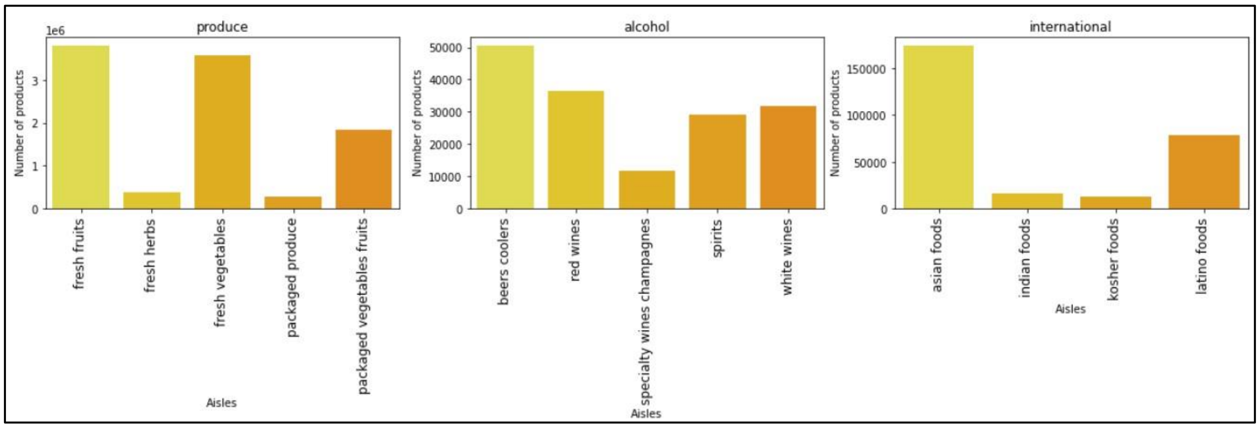


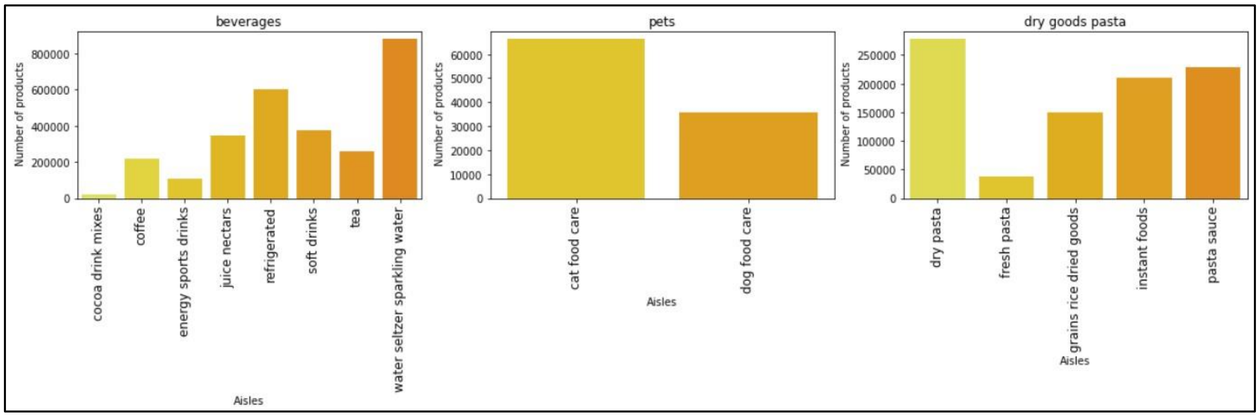
Now, let us look at the most important aisles in each department based on the number of products.

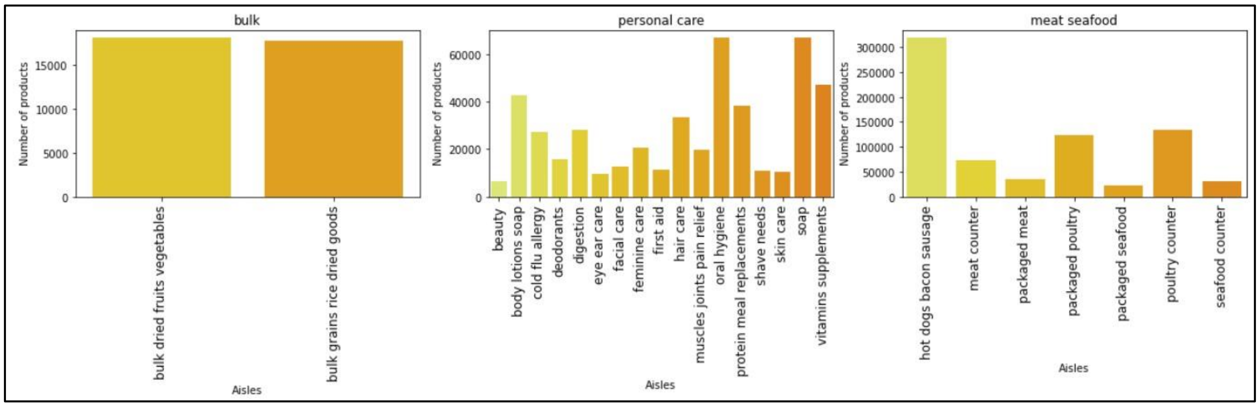


```
[32] fig,ax = plt.subplots(7,3, figsize=(20,45), gridspec_kw =  dict(hspace=1.4))
     for i, j in enumerate(ax.flatten()):
        if(i < len(departments_df["department"])):
           data=order_products_df[order_products_df.department == departments_df.loc[i,"department"]].groupby(['aisle']).count()['product_id']
           g = sns.barplot(data.aisle, data.product_id ,palette="Wistia", ax=j)
           j.set_title('{}'.format(departments_df.loc[i,"department"]))
           j.set(xlabel = "Aisles", ylabel=" Number of products")
           g.set_xticklabels(labels = data.aisle,rotation=90, fontsize=12)
```

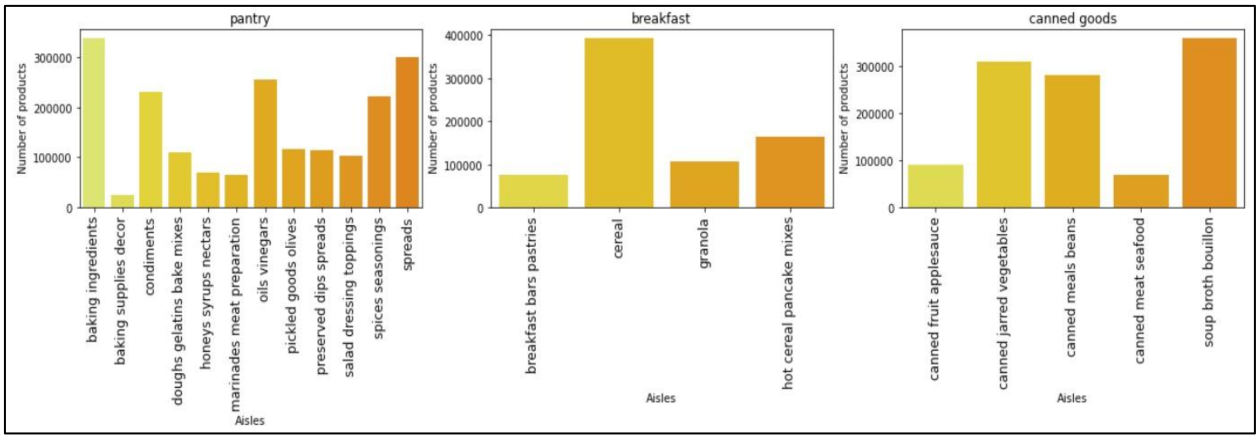




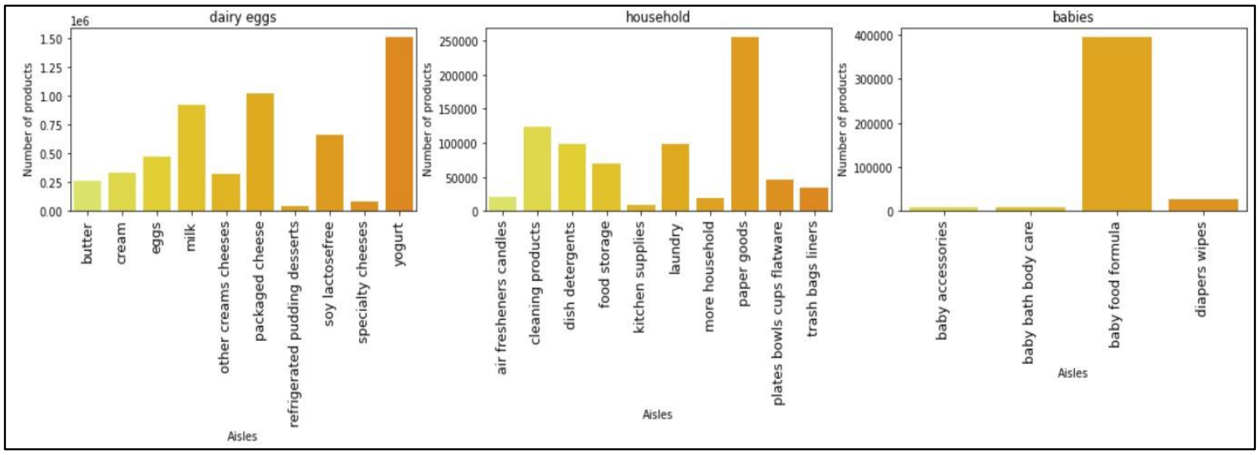




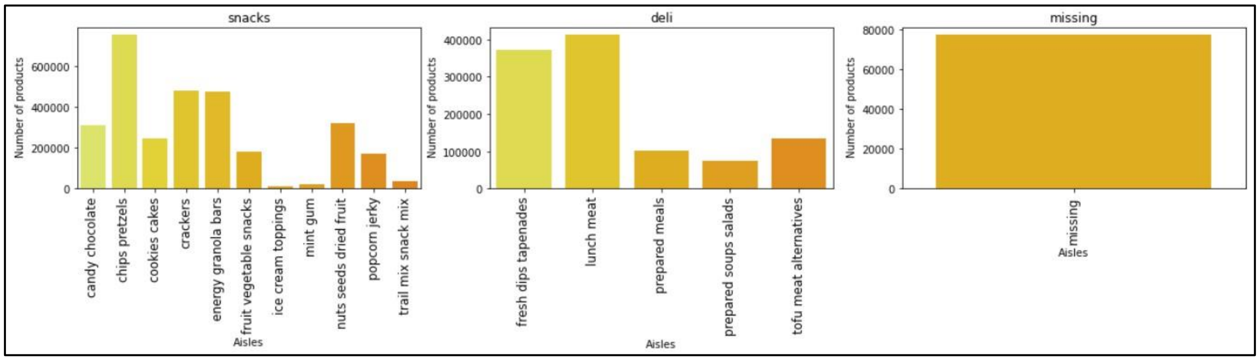
bulk
personal care
meat seafood
Number of products
Aisles
bulk dried fruits vegetables
bulk grains rice dried goods
beauty
body lotions soap
cold flu allergy
deodorants
digestion
eye ear care
facial care
feminine care
first aid
hair care
muscles joints pain relief
oral hygiene
protein meal replacements
shave needs
skin care
soap
vitamins supplements
hot dogs bacon sausage
meat counter
packaged meat
packaged poultry
packaged seafood
poultry counter
seafood counter



pantry
breakfast
canned goods
Number of products
Aisles
baking ingredients
baking supplies decor
condiments
doughs gelatins bake mixes
honeys syrups nectars
marinades meat preparation
oils vinegars
pickled goods olives
preserved dips spreads
salad dressing toppings
spices seasonings
spreads
breakfast bars pastries
cereal
granola
hot cereal pancake mixes
canned fruit applesauce
canned jarred vegetables
canned meals beans
canned meat seafood
soup broth bouillon



dairy eggs
household
babies
1e6
Number of products
Aisles
butter
cream
eggs
milk
other creams cheeses
packaged cheese
refrigerated pudding desserts
soy lactosefree
specialty cheeses
yogurt
air fresheners candles
cleaning products
dish detergents
food storage
kitchen supplies
laundry
more household
paper goods
plates bowls cups flatware
trash bags liners
baby accessories
baby bath body care
baby food formula
diapers wipes

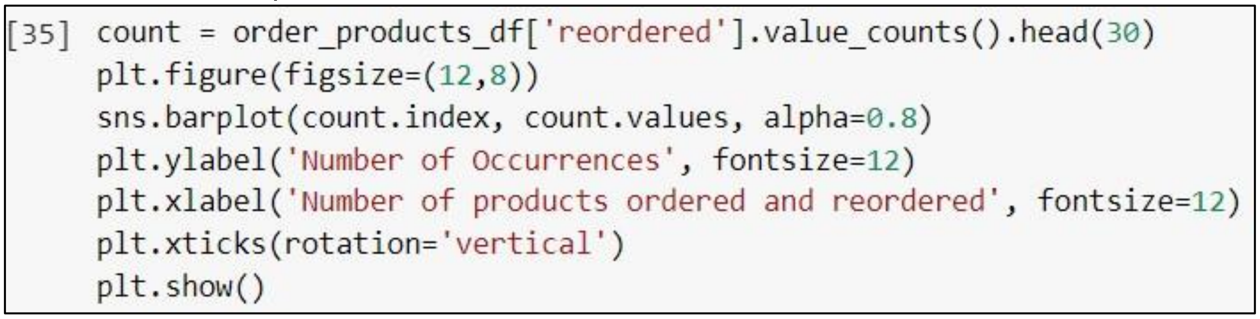


snacks
deli
missing
Number of products
Aisles
candy chocolate
chips pretzels
cookies cakes
crackers
energy granola bars
fruit vegetable snacks
ice cream toppings
mint gum
nuts seeds dried fruit
popcorn jerky
trail mix snack mix
fresh dips tapenades
lunch meat
prepared meals
prepared soups salads
tofu meat alternatives
missing

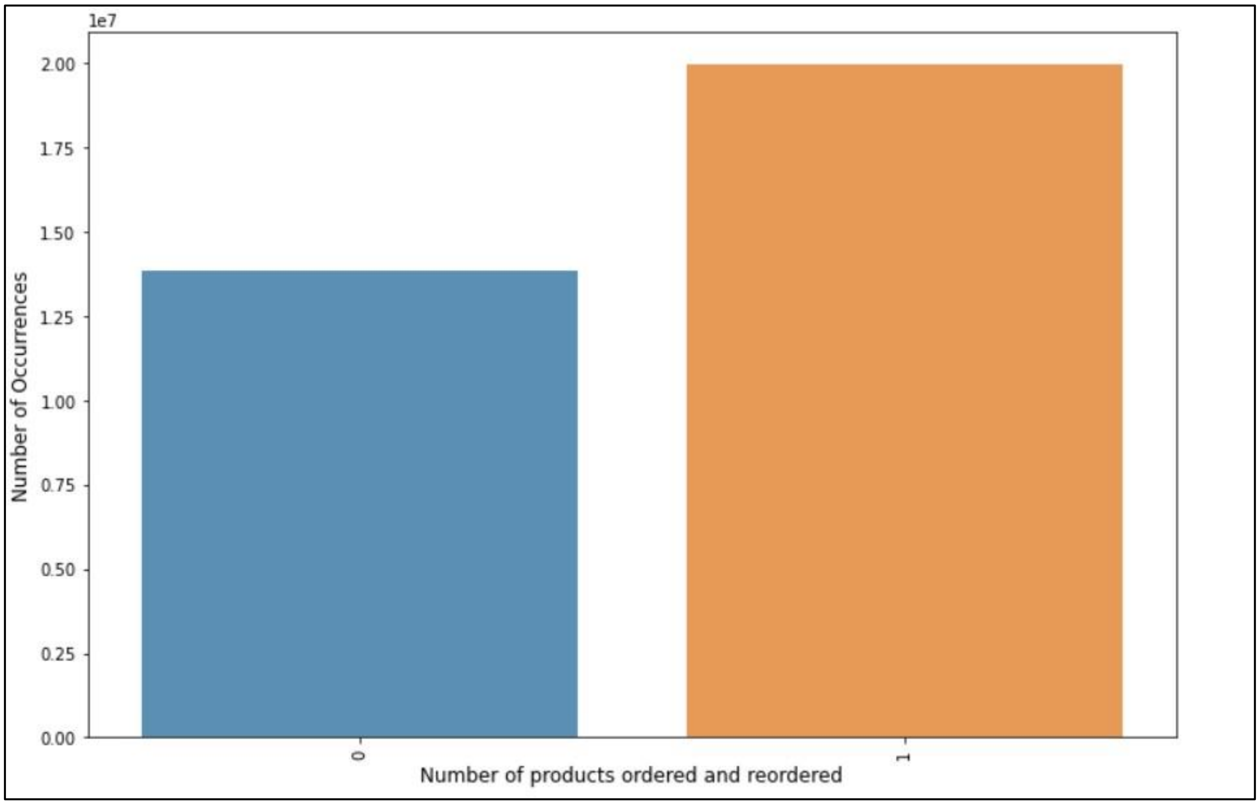
From the above plots, we can observe that ice-creams and frozen meals are most demanded in the frozen department. Then, bread is popular in the bakery section. Fresh fruits and fresh vegetables are in demand in the produce department. Beers coolers in the alcohol department. Asian foods are popular in the International department. Water seltzer and sparkling water in beverages. Cat food care is more in demand than dog food care. Dry pasta is popular in dry goods pasta. In the personal care department, soap and oral hygiene are in demand. In the meat and seafood department, hot dogs bacon sausage are in demand. Then, in the pantry, baking ingredients are in demand. For breakfast, cereal and in canned goods, soup broth bouillon. Also, in the dairy eggs department, yogurt is highly in demand. In the household department, paper goods are in demand. In the babies department, baby food formula is more in demand. Then in snacks, chips pretzels are in demand, and in deli lunch, meat is popular.

Now, let us have a look at the reordering factor at Instacart. Let us look at visualizations that give us some insight into the reordering factor.

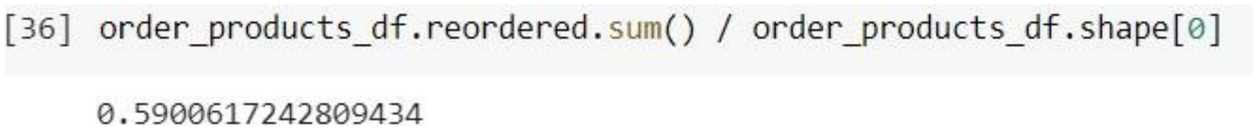
First, let us look at the number of occurrences for ordered and reordered products. In our dataset, the occurrences are represented using Boolean values 0 and 1. 0 represents the first order, and 1 represents reorder.



```
[35] count = order_products_df['reordered'].value_counts().head(30)
     plt.figure(figsize=(12,8))
     sns.barplot(count.index, count.values, alpha=0.8)
     plt.ylabel('Number of Occurrences', fontsize=12)
     plt.xlabel('Number of products ordered and reordered', fontsize=12)
     plt.xticks(rotation='vertical')
     plt.show()
```



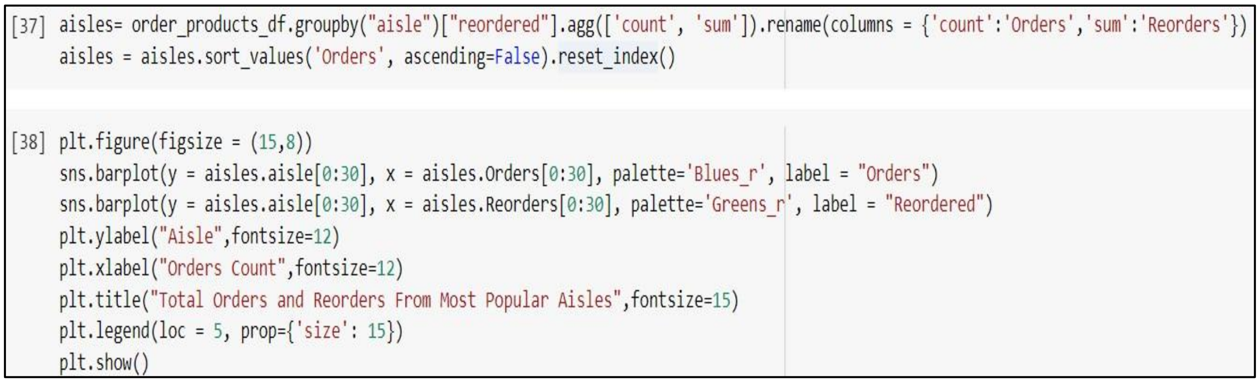
From the above barplot, we can see that the number of occurrences of products for 1 is more. That means the maximum number of products are reordered at Instacart.



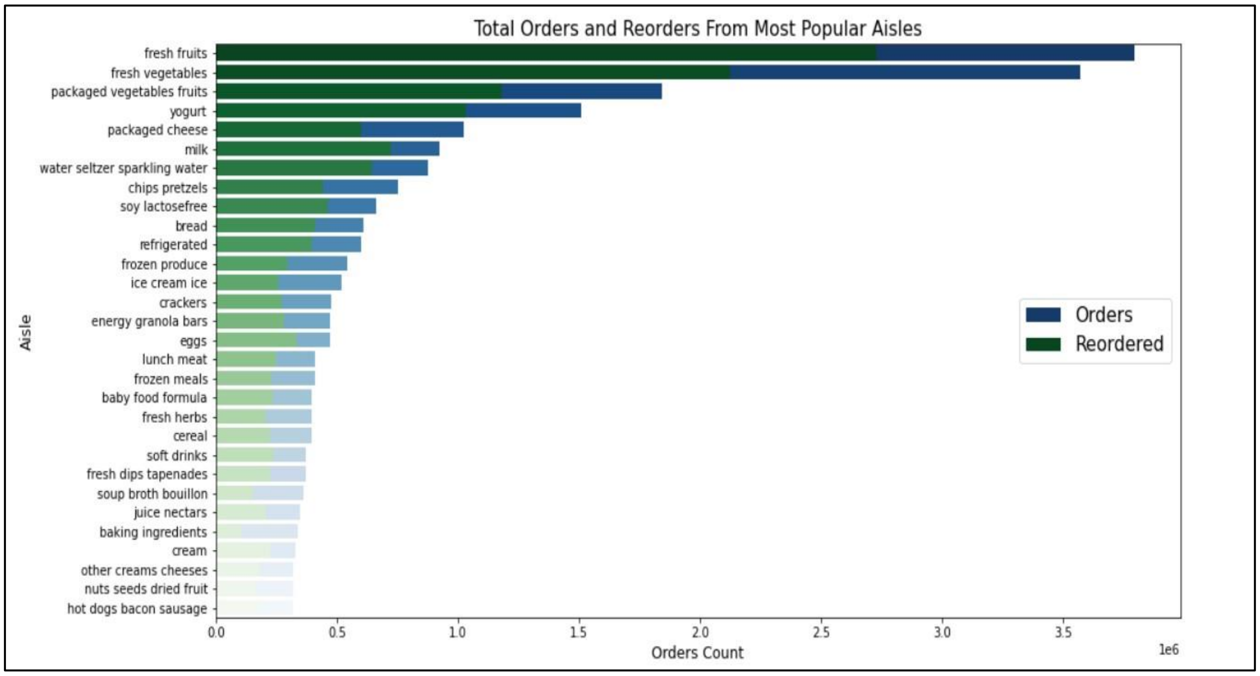
```
[36] order_products_df.reordered.sum() / order_products_df.shape[0]

     0.5900617242809434
```

From the above result, we can see that approximately 59% of the products at Instacart are reordered.

Now, let us have a look at the total orders and reorder at Instacart for different aisles.

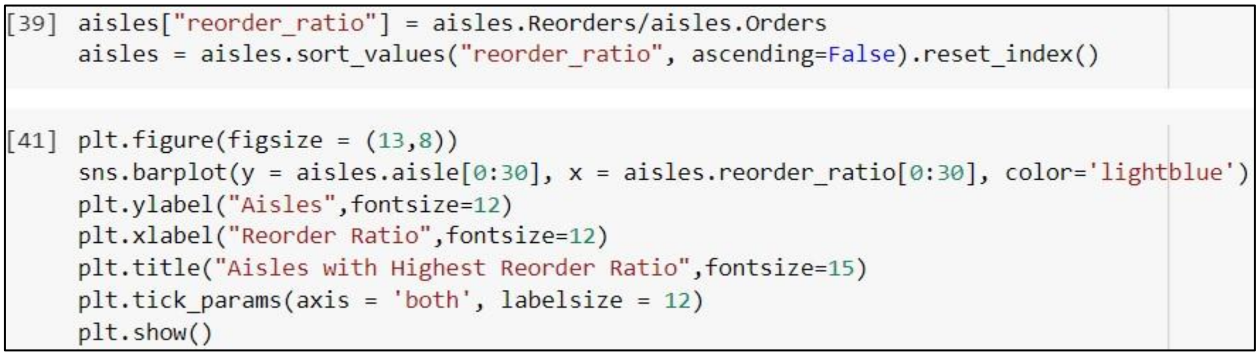


```
[37] aisles= order_products_df.groupby("aisle")["reordered"].agg(['count', 'sum']).rename(columns = {'count':'Orders','sum':'Reorders'})
     aisles = aisles.sort_values('Orders', ascending=False).reset_index()

[38] plt.figure(figsize = (15,8))
     sns.barplot(y = aisles.aisle[0:30], x = aisles.Orders[0:30], palette='Blues_r', label = "Orders")
     sns.barplot(y = aisles.aisle[0:30], x = aisles.Reorders[0:30], palette='Greens_r', label = "Reordered")
     plt.ylabel("Aisle",fontsize=12)
     plt.xlabel("Orders Count",fontsize=12)
     plt.title("Total Orders and Reorders From Most Popular Aisles",fontsize=15)
     plt.legend(loc = 5, prop={'size': 15})
     plt.show()
```

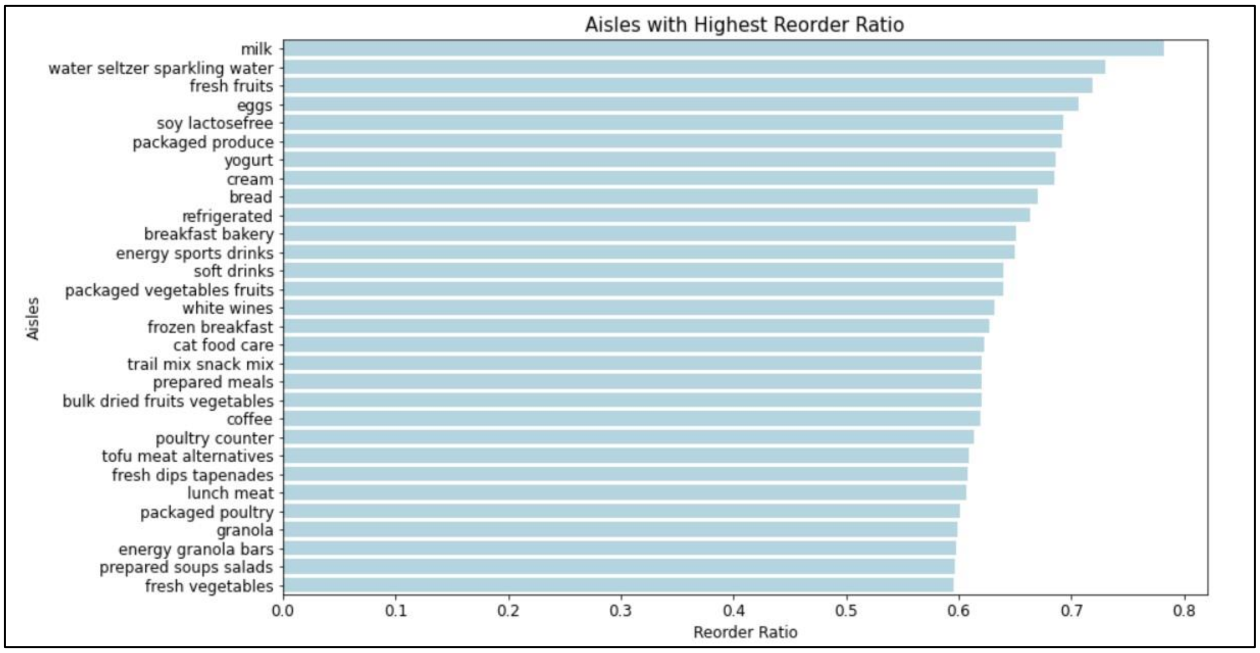


From the above visualization, we can see the total orders and reorders from fresh fruits and fresh vegetables aisles, followed by packaged vegetables, fruits, and yogurt.

Now, let us have a look at the highest and the lowest reorder ratio according to the aisles at Instacart.



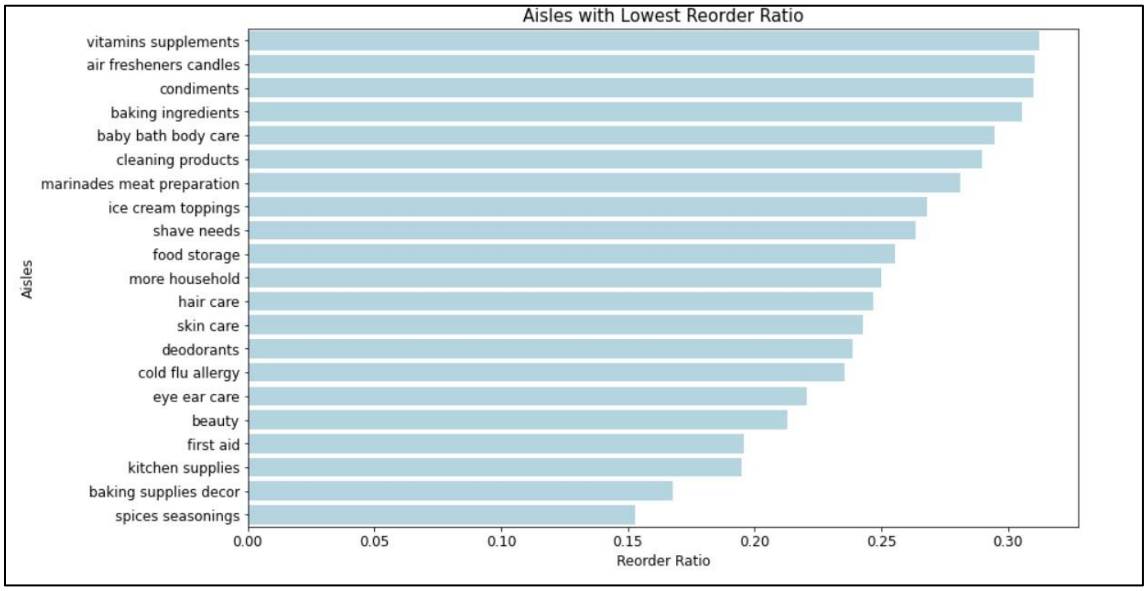
```
[39] aisles["reorder_ratio"] = aisles.Reorders/aisles.Orders
     aisles = aisles.sort_values("reorder_ratio", ascending=False).reset_index()

[41] plt.figure(figsize = (13,8))
     sns.barplot(y = aisles.aisle[0:30], x = aisles.reorder_ratio[0:30], color='lightblue')
     plt.ylabel("Aisles",fontsize=12)
     plt.xlabel("Reorder Ratio",fontsize=12)
     plt.title("Aisles with Highest Reorder Ratio",fontsize=15)
     plt.tick_params(axis = 'both', labelsize = 12)
     plt.show()
```



As per the above visualization, Milk, water seltzer sparkling water, fresh fruits, and eggs are the Aisles from which customers reorder the most.

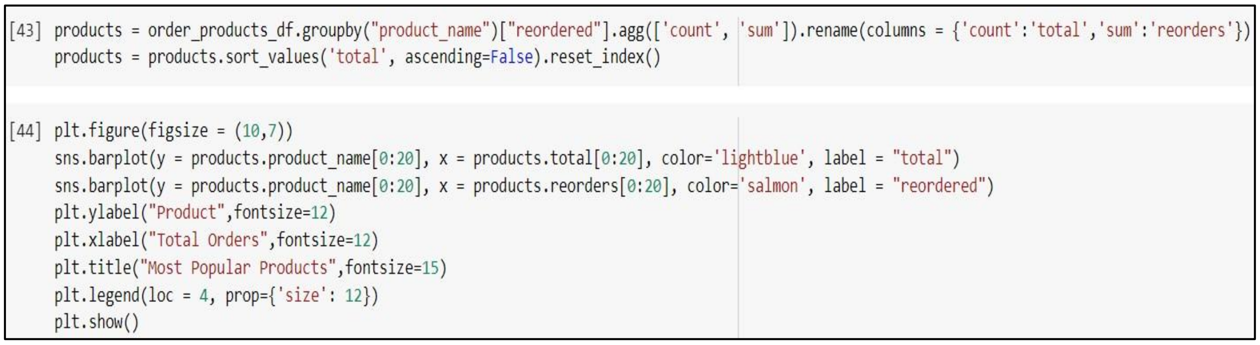
Now, let us take a look at the lowest reorder ratio.

```
plt.figure(figsize = (13,8))
sns.barplot(y = aisles.aisle[-21:], x = aisles.reorder_ratio[-21:], color='lightblue')
plt.ylabel("Aisles",fontsize=12)
plt.xlabel("Reorder Ratio",fontsize=12)
plt.title("Aisles with Lowest Reorder Ratio",fontsize=15)
plt.tick_params(axis = 'both', labelsize = 12)
plt.show()
```

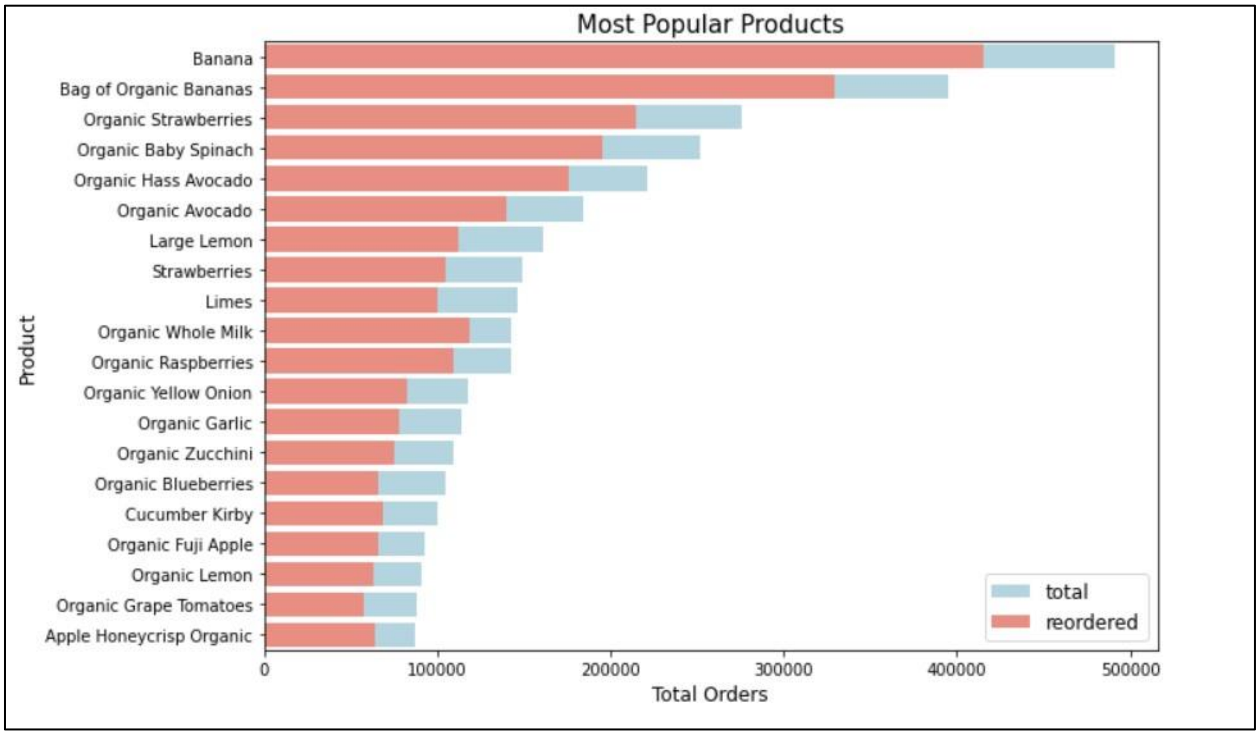


From the above visualization, we can see that vitamin supplements, air fresheners, candles, condiments, baking ingredients are Aisles with the lowest reorder ratio. It means that costumers order these kinds of products as per the need and not on a regular basis like fruits and vegetables.

Now, let us take a look at the most popular products that customers regularly reorder.

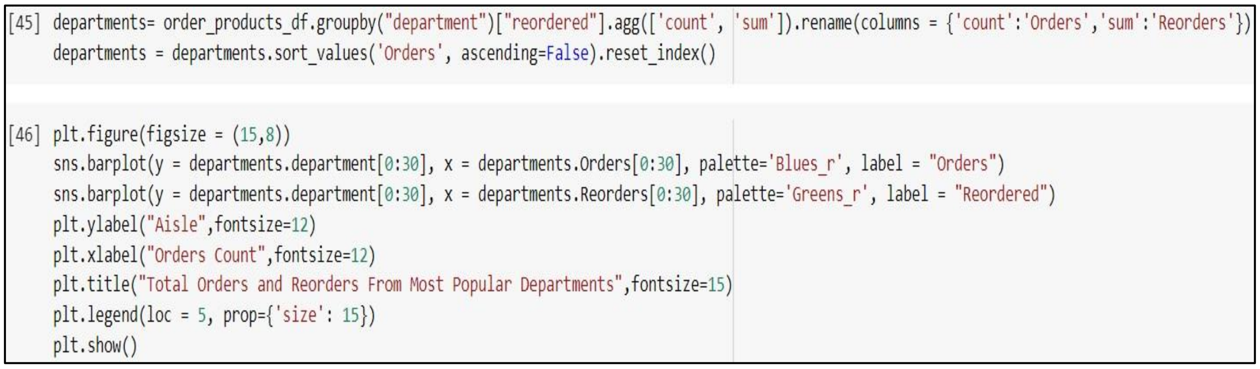


```
[43] products = order_products_df.groupby("product_name")["reordered"].agg(['count', 'sum']).rename(columns = {'count':'total','sum':'reorders'})
     products = products.sort_values('total', ascending=False).reset_index()

[44] plt.figure(figsize = (10,7))
     sns.barplot(y = products.product_name[0:20], x = products.total[0:20], color='lightblue', label = "total")
     sns.barplot(y = products.product_name[0:20], x = products.reorders[0:20], color='salmon', label = "reordered")
     plt.ylabel("Product",fontsize=12)
     plt.xlabel("Total Orders",fontsize=12)
     plt.title("Most Popular Products",fontsize=15)
     plt.legend(loc = 4, prop={'size': 12})
     plt.show()
```

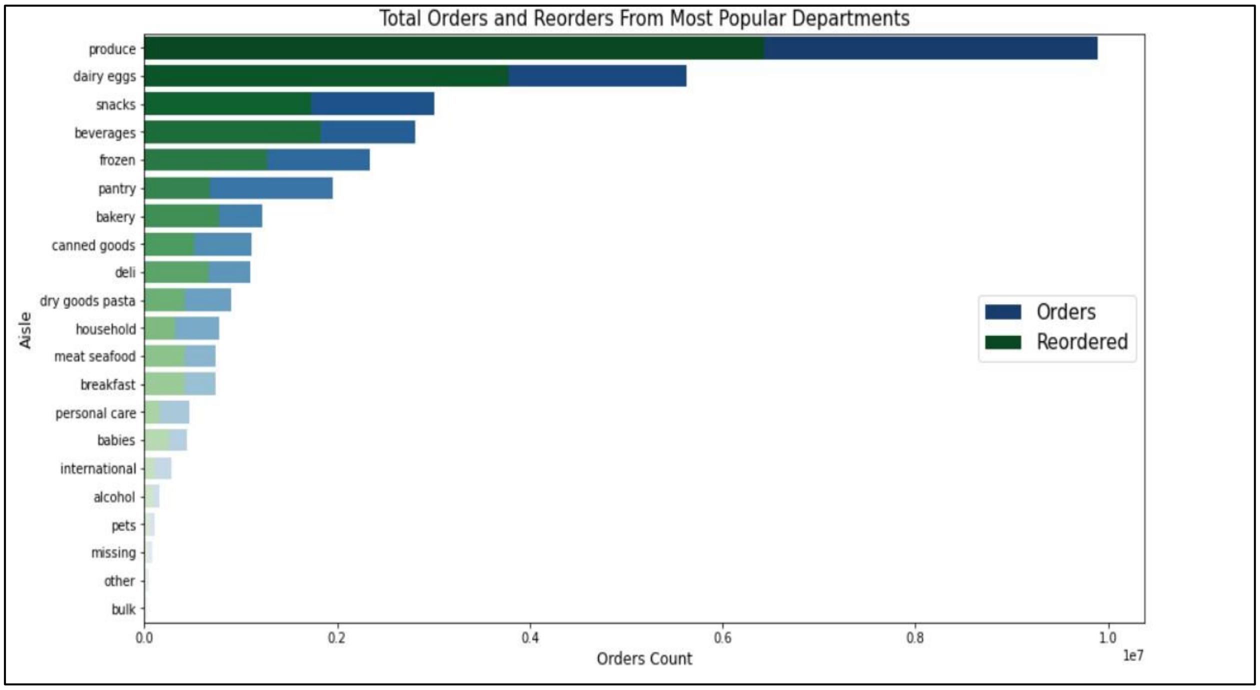


From the visualization, we can see that Bananas, Organic Bananas, Organic Strawberries, Organic Baby Spinach, Organic Hass Avocado, Limes, and Organic Whole Milk are the regularly reorder products. Also, we can see that a lot of products ordered are Organic. So, we will visualize a trend for Organic products as well.

Now, let us have a look at the total number of orders and reorders based on the departments.

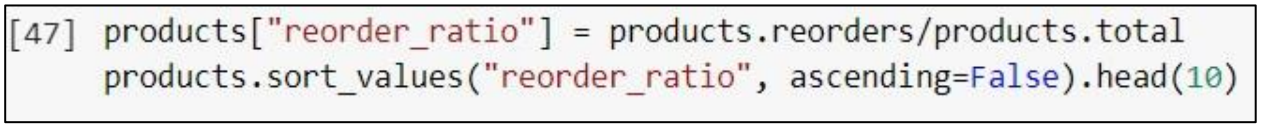


```
[45] departments= order_products_df.groupby("department")["reordered"].agg(['count', 'sum']).rename(columns = {'count':'Orders','sum':'Reorders'})
     departments = departments.sort_values('Orders', ascending=False).reset_index()

[46] plt.figure(figsize = (15,8))
     sns.barplot(y = departments.department[0:30], x = departments.Orders[0:30], palette='Blues_r', label = "Orders")
     sns.barplot(y = departments.department[0:30], x = departments.Reorders[0:30], palette='Greens_r', label = "Reordered")
     plt.ylabel("Aisle",fontsize=12)
     plt.xlabel("Orders Count",fontsize=12)
     plt.title("Total Orders and Reorders From Most Popular Departments",fontsize=15)
     plt.legend(loc = 5, prop={'size': 15})
     plt.show()
```

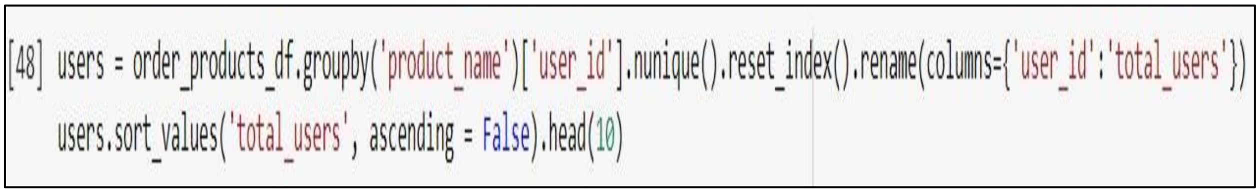


From the above visualization, we can clearly see that produce and dairy eggs are the most popular departments from which the products have been reordered.

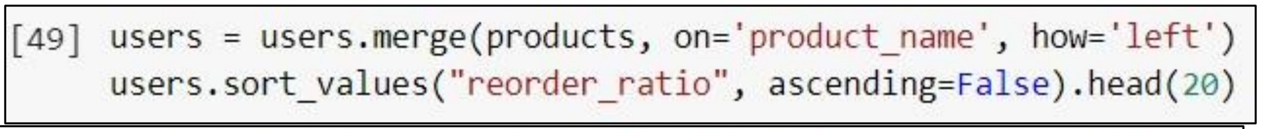
Now, let us have a look at the unique users at Instacart and the orders related to those users.



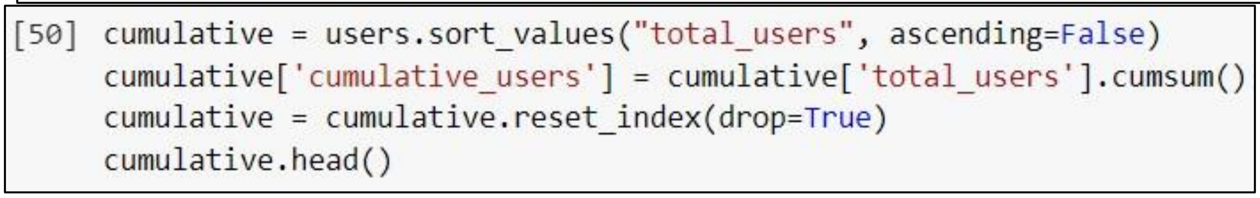
```
[47] products["reorder_ratio"] = products.reorders/products.total
     products.sort_values("reorder_ratio", ascending=False).head(10)
```



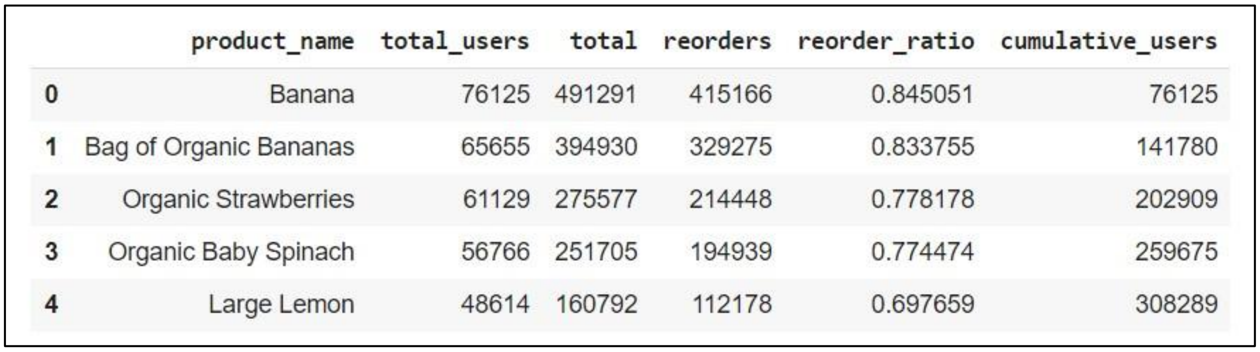
```
[48] users = order_products_df.groupby('product_name')['user_id'].nunique().reset_index().rename(columns={'user_id':'total_users'})
     users.sort_values('total_users', ascending = False).head(10)
```



```
[49] users = users.merge(products, on='product_name', how='left')
     users.sort_values("reorder_ratio", ascending=False).head(20)
```

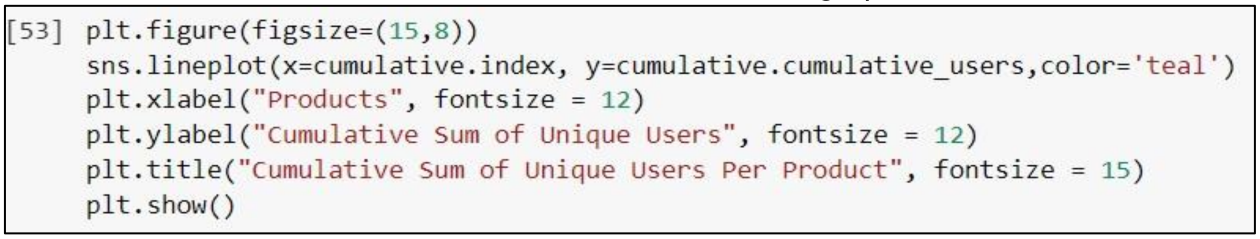


```
[50] cumulative = users.sort_values("total_users", ascending=False)
     cumulative['cumulative_users'] = cumulative['total_users'].cumsum()
     cumulative = cumulative.reset_index(drop=True)
     cumulative.head()
```

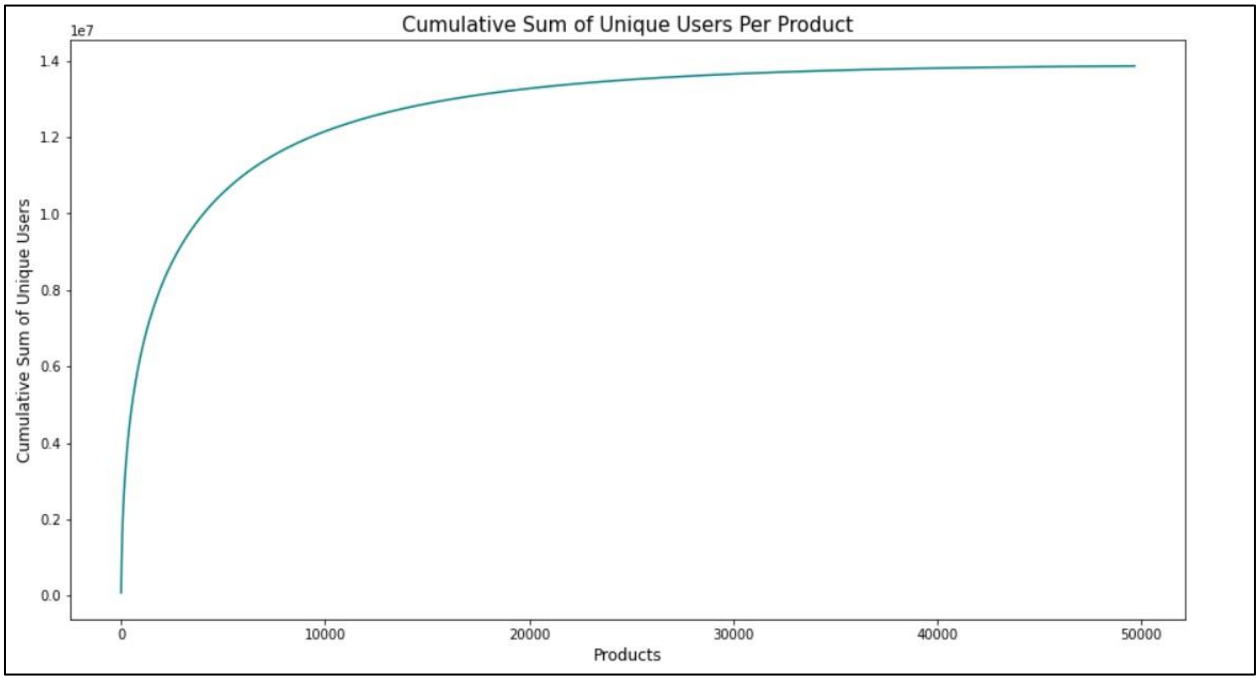


	product_name	total_users	total	reorders	reorder_ratio	cumulative_users
0	Banana	76125	491291	415166	0.845051	76125
1	Bag of Organic Bananas	65655	394930	329275	0.833755	141780
2	Organic Strawberries	61129	275577	214448	0.778178	202909
3	Organic Baby Spinach	56766	251705	194939	0.774474	259675
4	Large Lemon	48614	160792	112178	0.697659	308289

So, here we have a list of all the cumulative users associated with the products and the reorder ratio. Let us visualize those cumulative users in a graph.

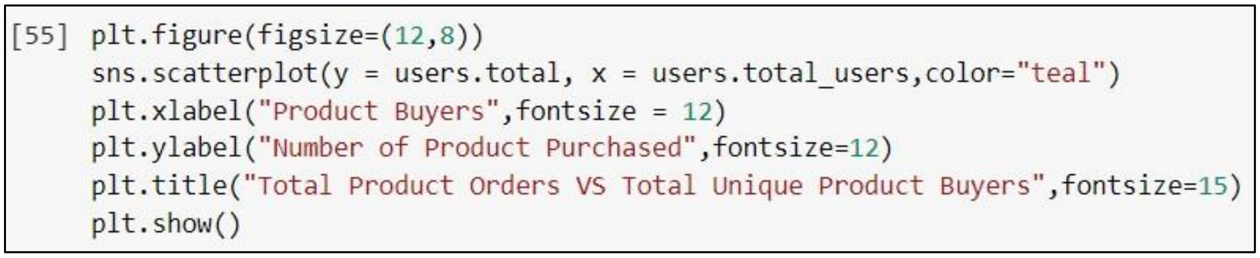


```
[53] plt.figure(figsize=(15,8))
     sns.lineplot(x=cumulative.index, y=cumulative.cumulative_users,color='teal')
     plt.xlabel("Products", fontsize = 12)
     plt.ylabel("Cumulative Sum of Unique Users", fontsize = 12)
     plt.title("Cumulative Sum of Unique Users Per Product", fontsize = 15)
     plt.show()
```

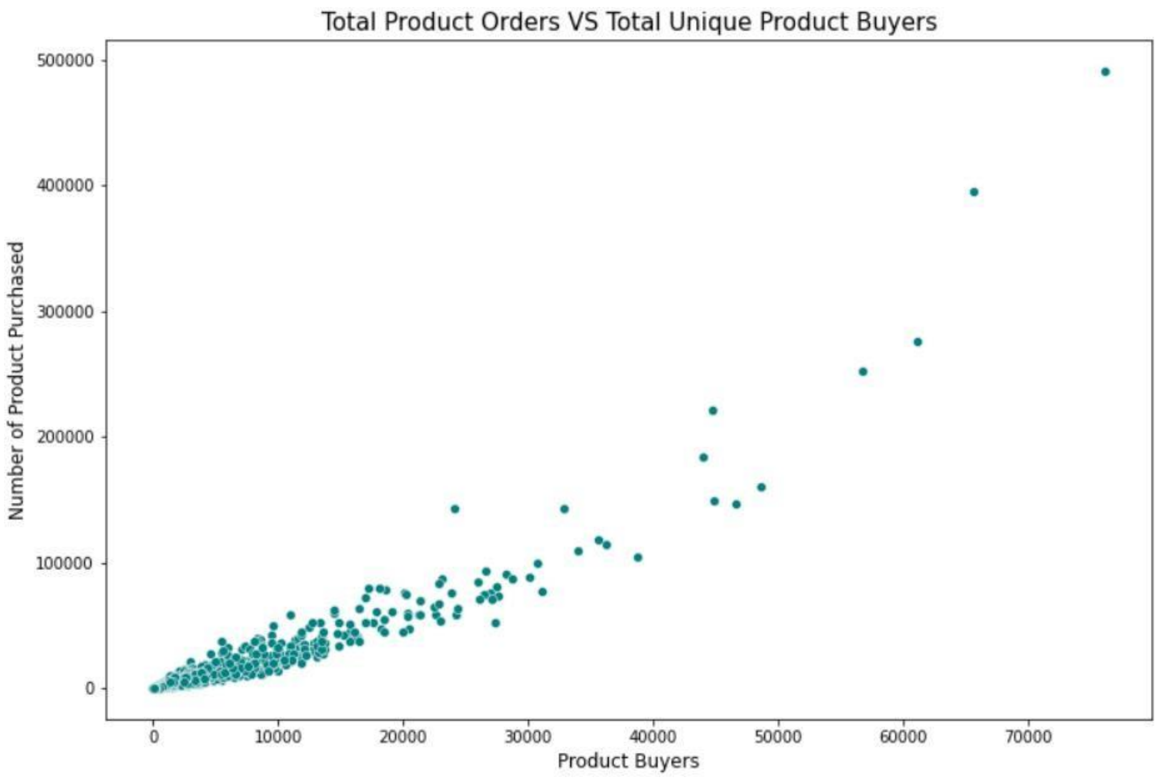


From the above graph, we can see that the cumulative sum of unique users vs. the products grows exponentially at first and then comes to a constant trend later.

Let us have a look at the below scatterplot.

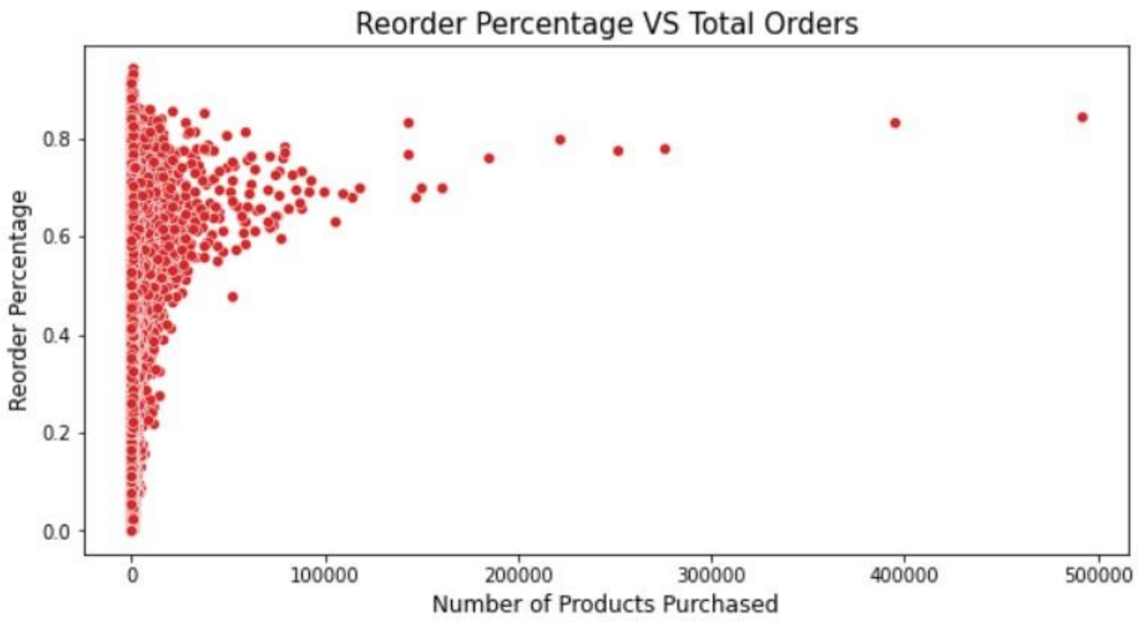


```
[55]  plt.figure(figsize=(12,8))
      sns.scatterplot(y = users.total, x = users.total_users,color="teal")
      plt.xlabel("Product Buyers",fontsize = 12)
      plt.ylabel("Number of Product Purchased",fontsize=12)
      plt.title("Total Product Orders VS Total Unique Product Buyers",fontsize=15)
      plt.show()
```



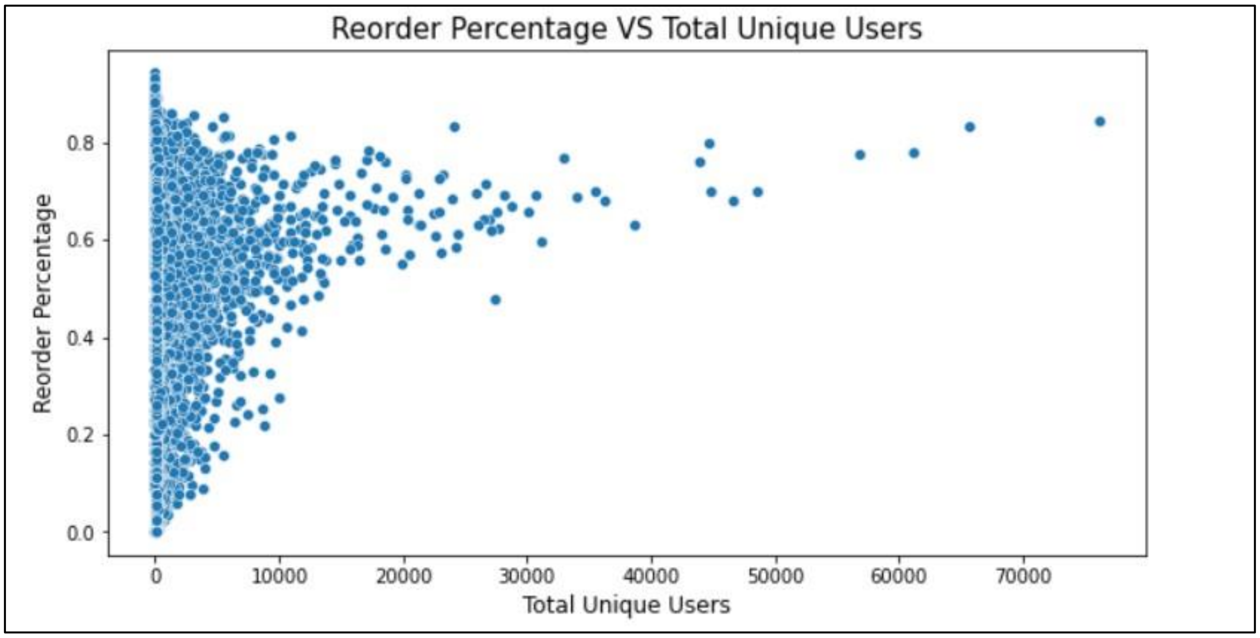
From the above scatterplot, we can see that there is a linear increase in the relationship between the products purchased and the product buyers.

```
[56] plt.figure(figsize=(10,5))
     sns.scatterplot(x = users.total, y = users.reorder_ratio, color = color[3])
     plt.xlabel("Number of Products Purchased",fontsize = 12)
     plt.ylabel("Reorder Percentage",fontsize = 12)
     plt.title("Reorder Percentage VS Total Orders",fontsize = 15)
     plt.show()
```



From the above scatterplot, we can see that for 0 to 30000 number of products purchased and the reorder percentage is varying between 0 to 100 percent. However, for the number of products greater than 30000, the reorder percentage is observed to be greater than 60%.

```
[58] plt.figure(figsize=(10,5))
     sns.scatterplot(x = users.total_users, y = users.reorder_ratio, color = color[0])
     plt.xlabel("Total Unique Users",fontsize = 12)
     plt.ylabel("Reorder Percentage",fontsize = 12)
     plt.title("Reorder Percentage VS Total Unique Users",fontsize = 15)
     plt.show()
```



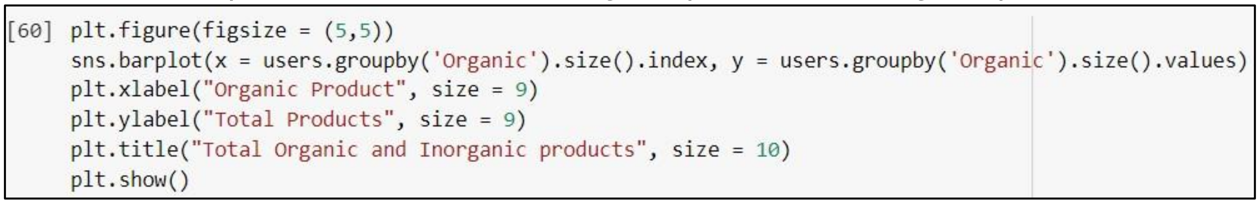
From this scatterplot, we can see that from 0 to 10000 total unique users, and the reorder percentage is varying from 0 to 100%. However, after 10000 unique users, the reorder percentage is constantly observed to be above 60%.

Now, let us have a look at the organic and inorganic products in our data.

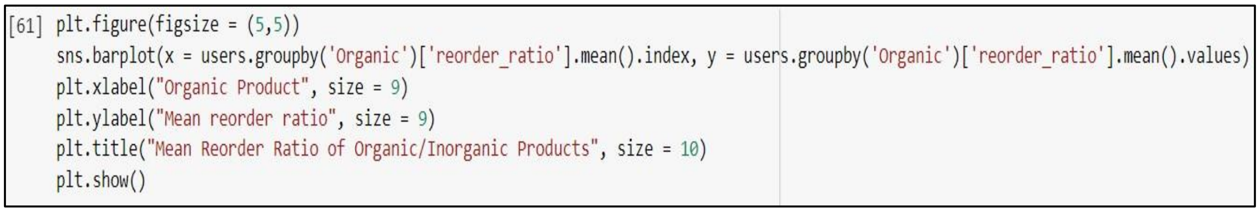
```
[59] users['Organic'] = users.product_name.str.contains("Organic")
     users.head()
```

	product_name	total_users	total	reorders	reorder_ratio	Organic
0	#2 Coffee Filters	534	799	265	0.331665	False
1	#2 Cone White Coffee Filters	11	14	3	0.214286	False
2	#2 Mechanical Pencils	9	9	0	0.000000	False
3	#4 Natural Brown Coffee Filters	157	247	90	0.364372	False
4	& Go! Hazelnut Spread + Pretzel Sticks	81	174	93	0.534483	False

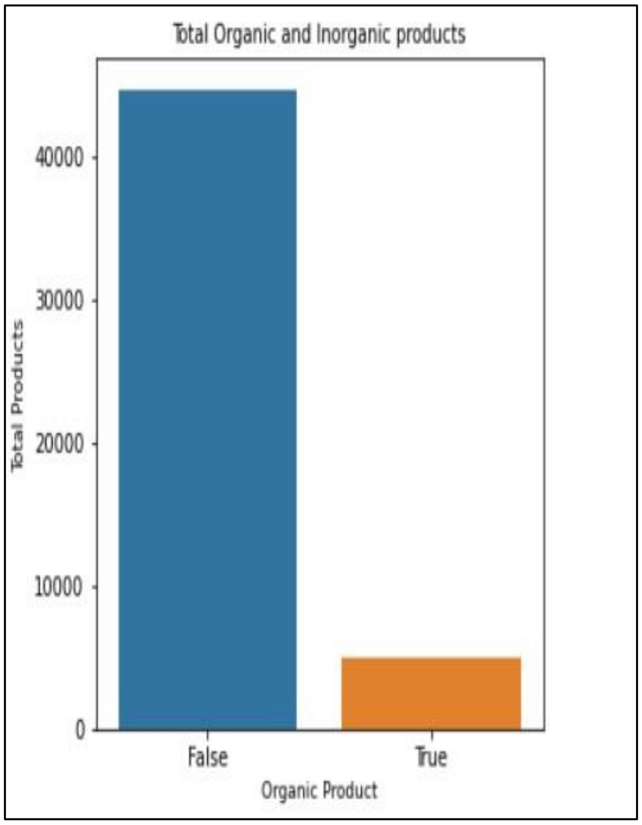
Now, let us compare the occurrences of organic products vs inorganic products.

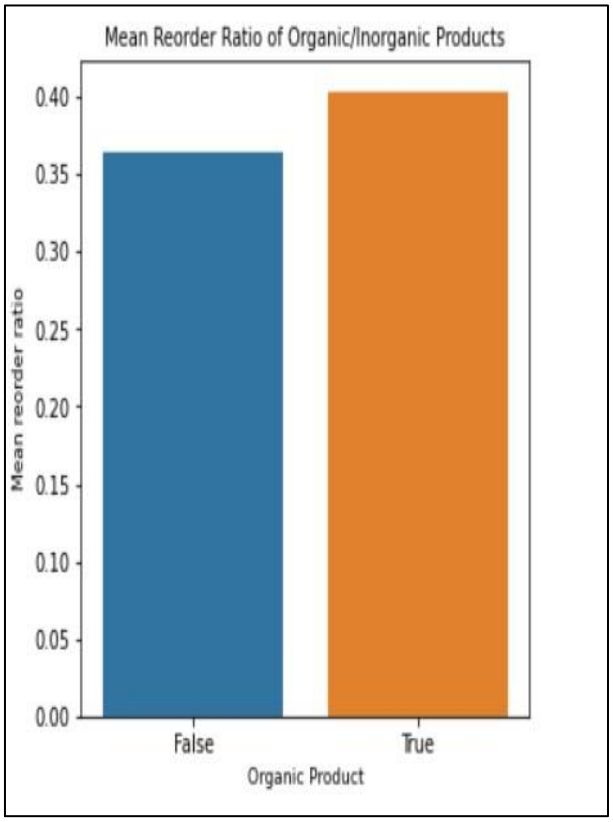


```
[60] plt.figure(figsize = (5,5))
     sns.barplot(x = users.groupby('Organic').size().index, y = users.groupby('Organic').size().values)
     plt.xlabel("Organic Product", size = 9)
     plt.ylabel("Total Products", size = 9)
     plt.title("Total Organic and Inorganic products", size = 10)
     plt.show()
```



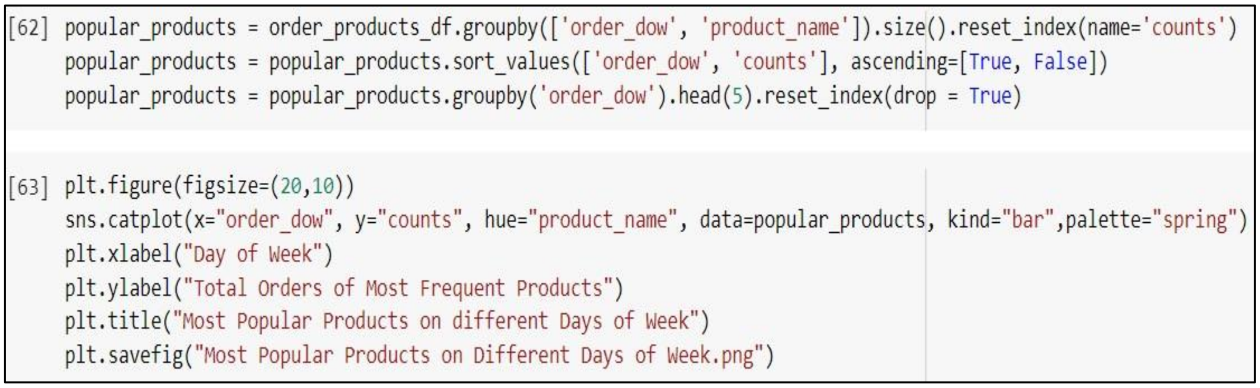
```
[61] plt.figure(figsize = (5,5))
     sns.barplot(x = users.groupby('Organic')['reorder_ratio'].mean().index, y = users.groupby('Organic')['reorder_ratio'].mean().values)
     plt.xlabel("Organic Product", size = 9)
     plt.ylabel("Mean reorder ratio", size = 9)
     plt.title("Mean Reorder Ratio of Organic/Inorganic Products", size = 10)
     plt.show()
```



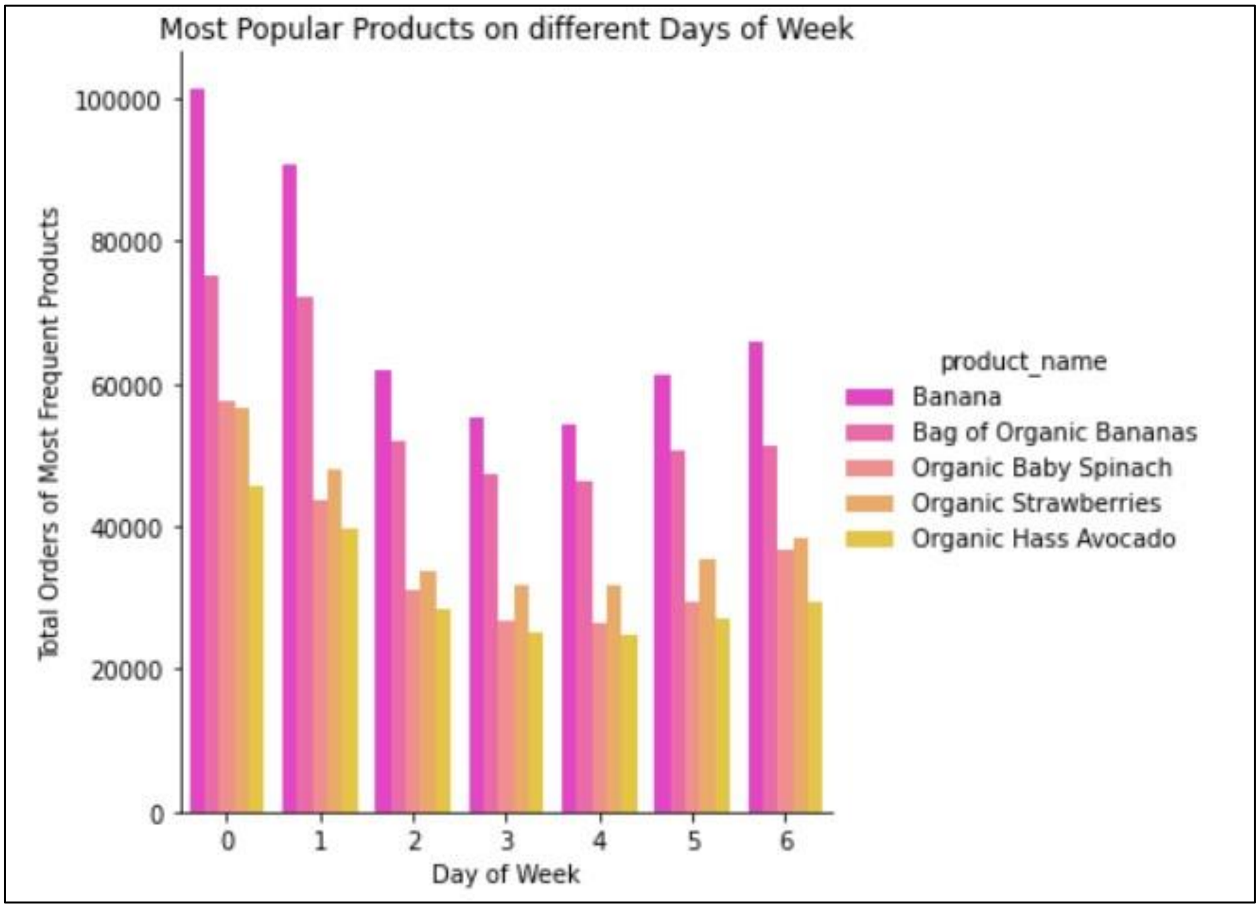


We can see that there is less number of organic products, but their Mean reorder percentage is high. This tells us that we should have more organic products in the store.

Now let us look at the most frequent products based on the day of the week.



```
[62] popular_products = order_products_df.groupby(['order_dow', 'product_name']).size().reset_index(name='counts')
     popular_products = popular_products.sort_values(['order_dow', 'counts'], ascending=[True, False])
     popular_products = popular_products.groupby('order_dow').head(5).reset_index(drop = True)

[63] plt.figure(figsize=(20,10))
     sns.catplot(x="order_dow", y="counts", hue="product_name", data=popular_products, kind="bar",palette="spring")
     plt.xlabel("Day of Week")
     plt.ylabel("Total Orders of Most Frequent Products")
     plt.title("Most Popular Products on different Days of Week")
     plt.savefig("Most Popular Products on Different Days of Week.png")
```



From the above graph, we can see that Bananas and Organic Bananas are the most frequent product on all days of the week.

6. ASSOCIATION RULE MINING

Now that we have analyzed our dataset and gathered insights for the data. Let us now have a look at how the data is actually associated with each other. For that, we will be using the Apriori algorithm to generate association rules. This will help us find out the association between different items in our dataset and how they are related.

Before implementing the Algorithm, let us briefly understand what association rules in Market Basket Analysis are and the terminologies related to it.

Association rule mining, as the name suggests, is a technique used to identify associations or relations between different items in our data. It is a technique to find a pattern of orders or transactions. Market Basket Analysis creates If-Then scenario rules; for example, if item A is purchased, then item B is likely to be purchased. For example, in a supermarket store, if a customer adds Milk to his cart, then he is more likely to add bread and butter as well. Or if a customer adds eggs and avocado, then he is more likely to add bread to the cart. The technique determines relationships of what products were purchased with which other product(s). These relationships are then used to build profiles containing If-Then rules of the items purchased.

In practice, the analysis is done on millions of transactions to identify the association between the items or products. This analysis uncovers the hidden purchasing patterns of the customers, and this data can then be used by the retailers to promote the recommended items and sell them.

We can observe this kind of implementation on e-commerce sites like Amazon, where they give you a suggestion like ((Amazon's customers who bought this product also bought these products). And then show you the suggested items.

So, the reason behind conducting the association rule mining on our data is to get the association patterns between different products in our dataset so that we can know which items or products the users or customers are more or less likely to purchase together. And then, once we generate these association rules, we can make product recommendations to the users based on the items they add to their cart for their next purchase.

Now, different statistical algorithms have been developed to implement association rule mining, and Apriori is one such Algorithm. This Algorithm is used with relational databases for frequent itemset mining and association rule learning. It uses a bottom-up approach where frequent items are extended one item at a time, and groups of candidates are tested against the available dataset. This process continues until no further extensions are found. The Apriori algorithm uses the concept of Support, Confidence, and Lift.

Let us have a look at all the terminologies used in the Algorithm:

Support: This shows how popular an item or a product is in our data. It calculates the percentage of orders that contain the item set.

The formula for support is given as **Support = Freq(X, Y)/N**

Confidence: This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. Given two items, X and Y, confidence measures the percentage of times that item Y is purchased, given that item X was purchased.

This is expressed as: **Confidence = Freq(X,Y)/Freq(X)**

Lift: Lift is said to measure the difference — measured in ratio — between the confidence of a rule and the expected confidence. This is expressed as:

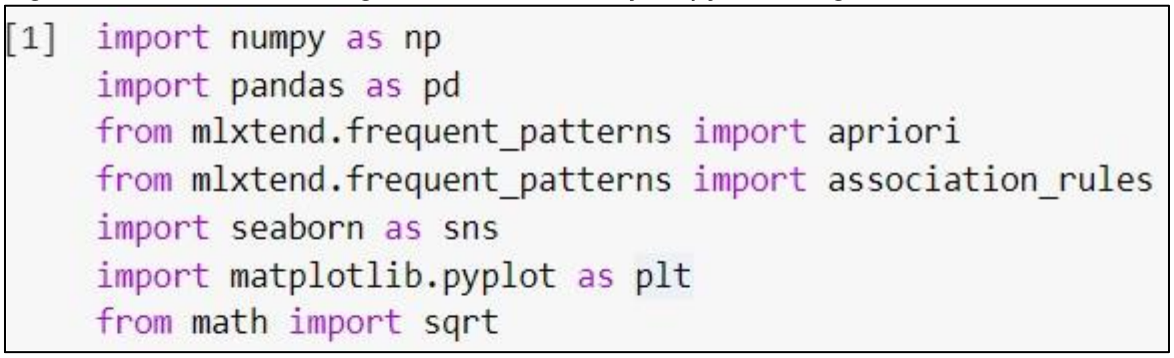
lift{X,Y} = lift{Y,X} = support{X,Y} / (support{X} * support{Y})

Important points to remember:

- Confidence values range from 0 to 1, where 0 indicates that Y is never purchased when X is purchased, and one indicates that Y is always purchased whenever X is purchased.
- Lift = 1; implies no relationship between X and Y (i.e., X and Y occur together only by chance)
- Lift > 1; implies that there is a positive relationship between X and Y (i.e., X and Y occur together more often than random)
- Lift < 1; implies that there is a negative relationship between X and Y (i.e., X and Y occur together less often than random)

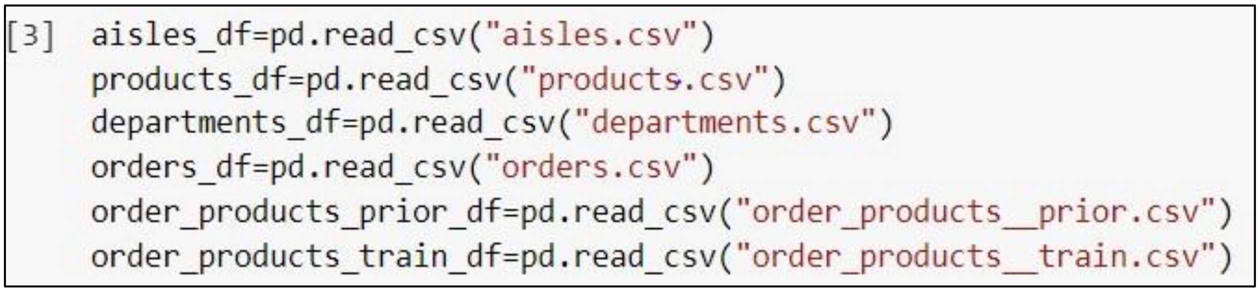
Let us have a look at the implementation of the Apriori algorithm in python.

First, we will import all the required libraries in python. For implementing the Apriori algorithm, we will be using the mlxtend library of python to generate the rules.



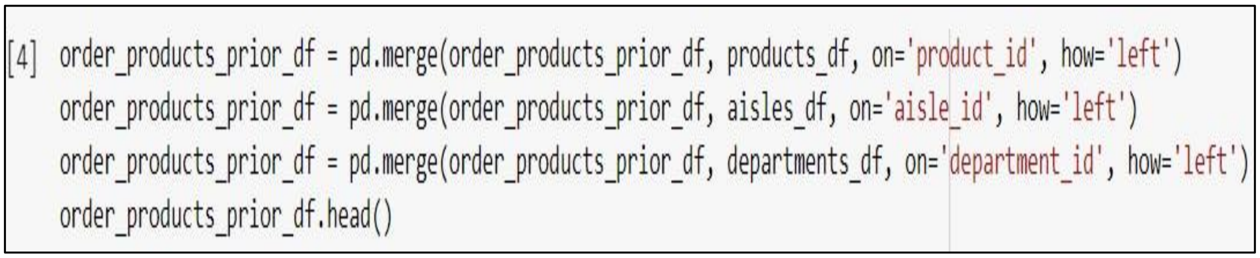
```
[1]  import numpy as np
     import pandas as pd
     from mlxtend.frequent_patterns import apriori
     from mlxtend.frequent_patterns import association_rules
     import seaborn as sns
     import matplotlib.pyplot as plt
     from math import sqrt
```

Now, let us load all the datasets into our Jupiter notebook and then convert them to dataframes.

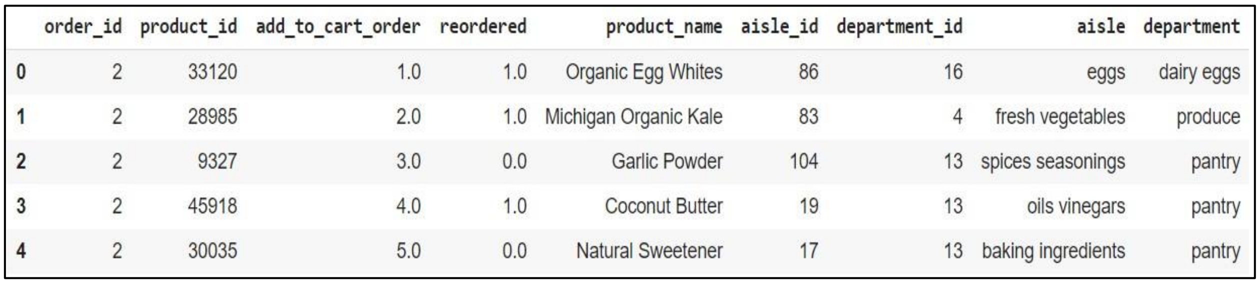


```
[3]  aisles_df=pd.read_csv("aisles.csv")
     products_df=pd.read_csv("products.csv")
     departments_df=pd.read_csv("departments.csv")
     orders_df=pd.read_csv("orders.csv")
     order_products_prior_df=pd.read_csv("order_products__prior.csv")
     order_products_train_df=pd.read_csv("order_products__train.csv")
```

Now, let us merge all the datasets together and put them into one single dataframe so that we will be able to work with all the attributes of the dataset.

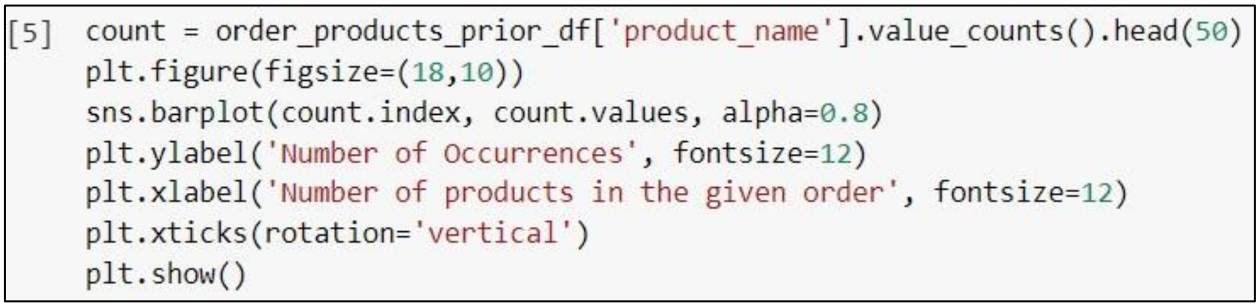


```
[4]  order_products_prior_df = pd.merge(order_products_prior_df, products_df, on='product_id', how='left')
     order_products_prior_df = pd.merge(order_products_prior_df, aisles_df, on='aisle_id', how='left')
     order_products_prior_df = pd.merge(order_products_prior_df, departments_df, on='department_id', how='left')
     order_products_prior_df.head()
```

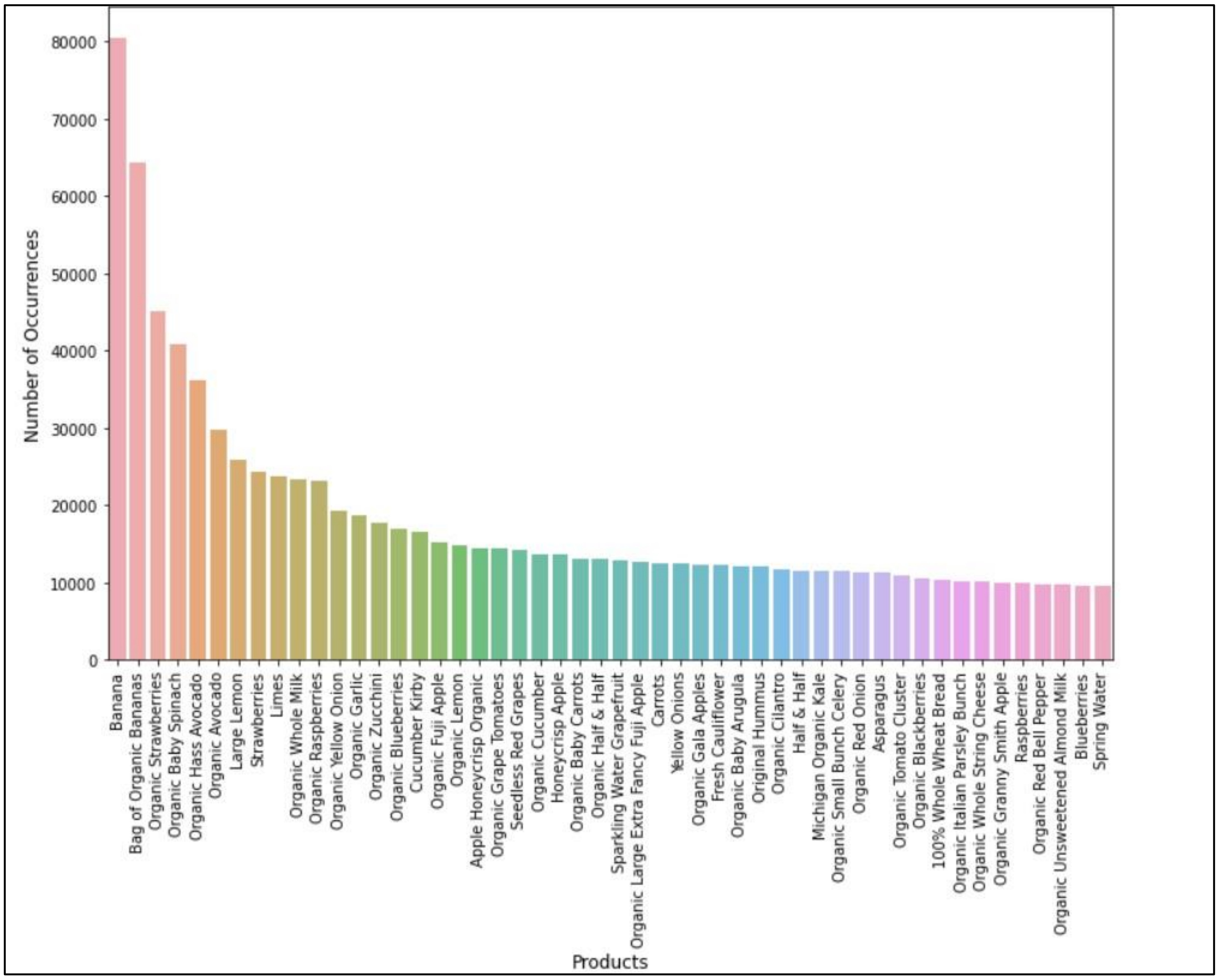


```
   order_id  product_id  add_to_cart_order  reordered          product_name  aisle_id  department_id               aisle  department
0         2       33120                1.0        1.0    Organic Egg Whites        86             16                eggs  dairy eggs
1         2       28985                2.0        1.0  Michigan Organic Kale        83              4    fresh vegetables     produce
2         2        9327                3.0        0.0         Garlic Powder       104             13   spices seasonings      pantry
3         2       45918                4.0        1.0        Coconut Butter        19             13        oils vinegars      pantry
4         2       30035                5.0        0.0     Natural Sweetener        17             13  baking ingredients      pantry
```

Now, let us have a look at the products in our data.

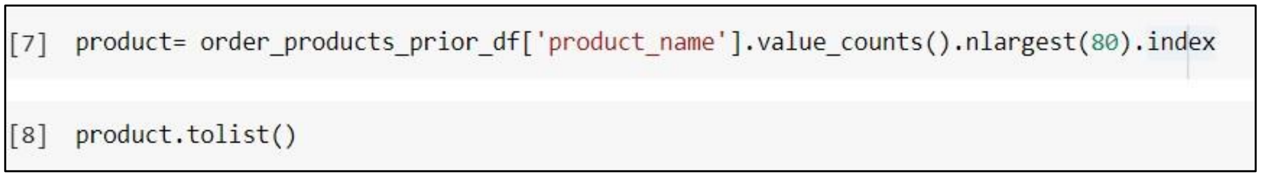


```
[5]  count = order_products_prior_df['product_name'].value_counts().head(50)
     plt.figure(figsize=(18,10))
     sns.barplot(count.index, count.values, alpha=0.8)
     plt.ylabel('Number of Occurrences', fontsize=12)
     plt.xlabel('Number of products in the given order', fontsize=12)
     plt.xticks(rotation='vertical')
     plt.show()
```



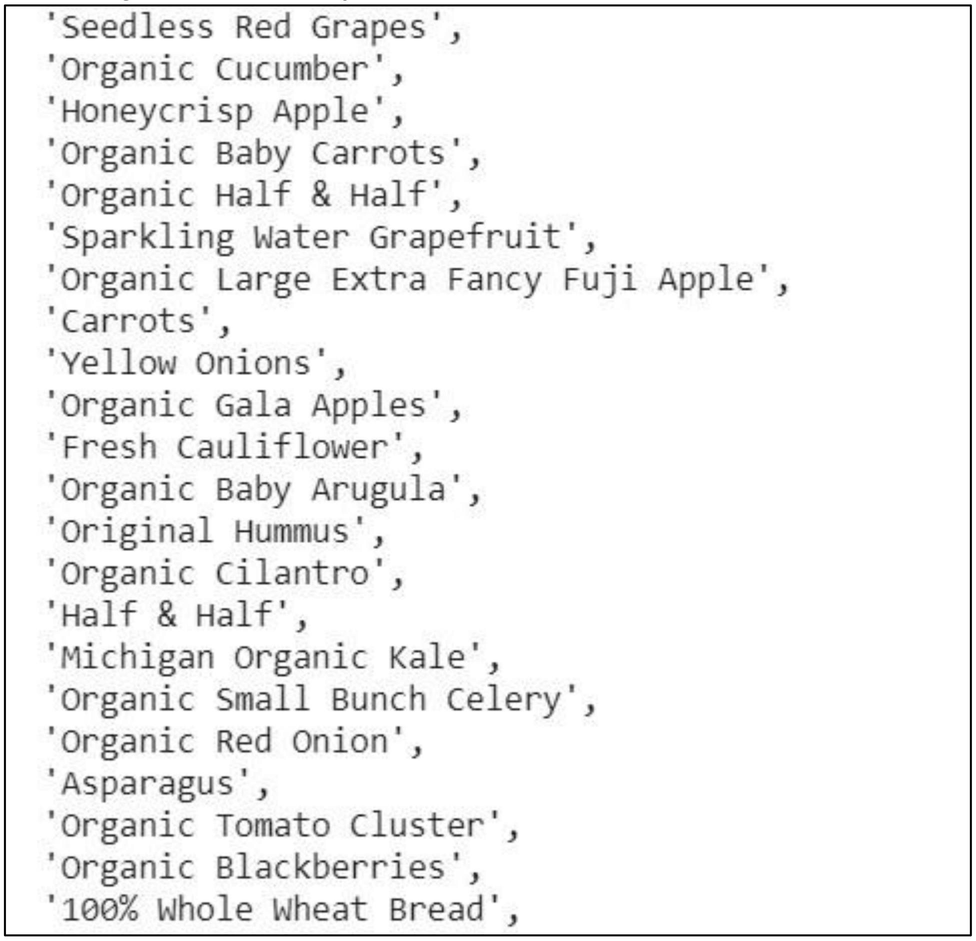
We can see that Bananas, Organic Bananas, Organic Strawberries, Organic Baby Spinach are the most frequently purchased products. Let us select just the top 80 products from our dataset since the original dataset is very large, and processing it will be time-consuming.

After that, let us convert our top 80 products to a list.



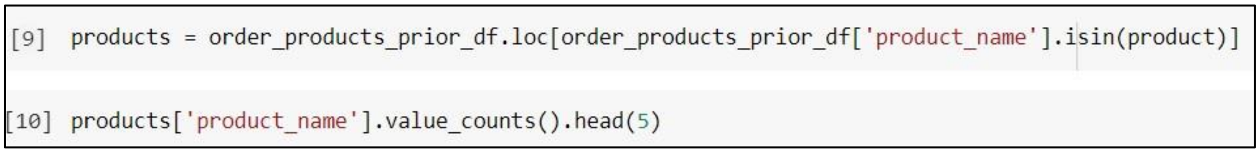
```
[7]  product= order_products_prior_df['product_name'].value_counts().nlargest(80).index

[8]  product.tolist()
```

Following is a list of our products.

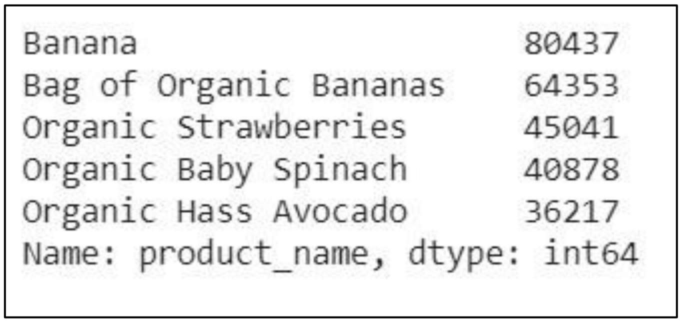


```
 'Seedless Red Grapes',
 'Organic Cucumber',
 'Honeycrisp Apple',
 'Organic Baby Carrots',
 'Organic Half & Half',
 'Sparkling Water Grapefruit',
 'Organic Large Extra Fancy Fuji Apple',
 'Carrots',
 'Yellow Onions',
 'Organic Gala Apples',
 'Fresh Cauliflower',
 'Organic Baby Arugula',
 'Original Hummus',
 'Organic Cilantro',
 'Half & Half',
 'Michigan Organic Kale',
 'Organic Small Bunch Celery',
 'Organic Red Onion',
 'Asparagus',
 'Organic Tomato Cluster',
 'Organic Blackberries',
 '100% Whole Wheat Bread',
```

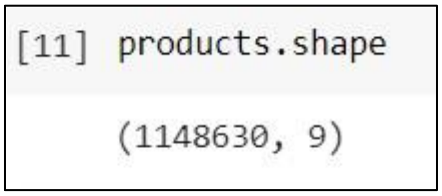
Then, let us get the frequency count of each product so that we know how many times the product has been purchased by the customers.



```
[9]  products = order_products_prior_df.loc[order_products_prior_df['product_name'].isin(product)]

[10] products['product_name'].value_counts().head(5)
```

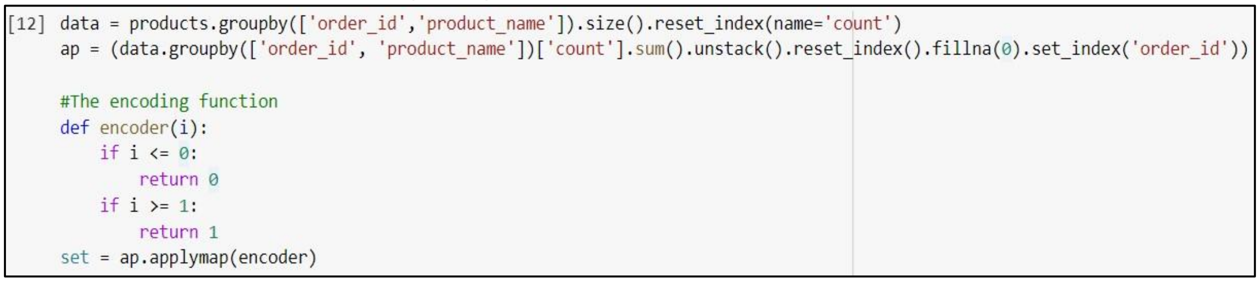


```
Banana                      80437
Bag of Organic Bananas      64353
Organic Strawberries        45041
Organic Baby Spinach        40878
Organic Hass Avocado        36217
Name: product_name, dtype: int64
```



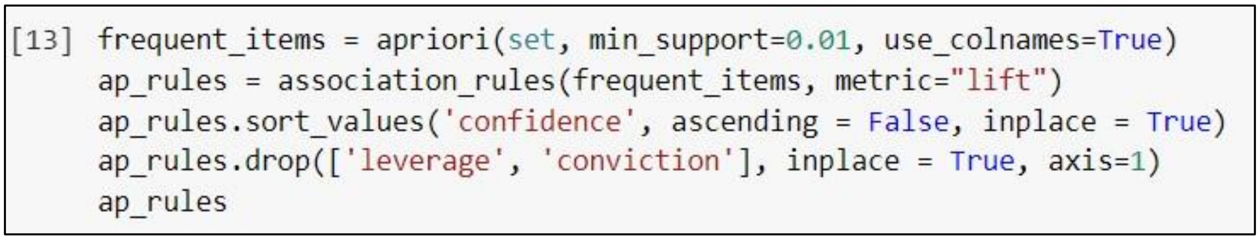
```
[11] products.shape

     (1148630, 9)
```

Now, we have 1148630 products and nine columns. Now let us apply the apriori Algorithm to our products. Let us apply the encoder to our data, so any positive values are converted to one, and anything less the 0 is set to 0.

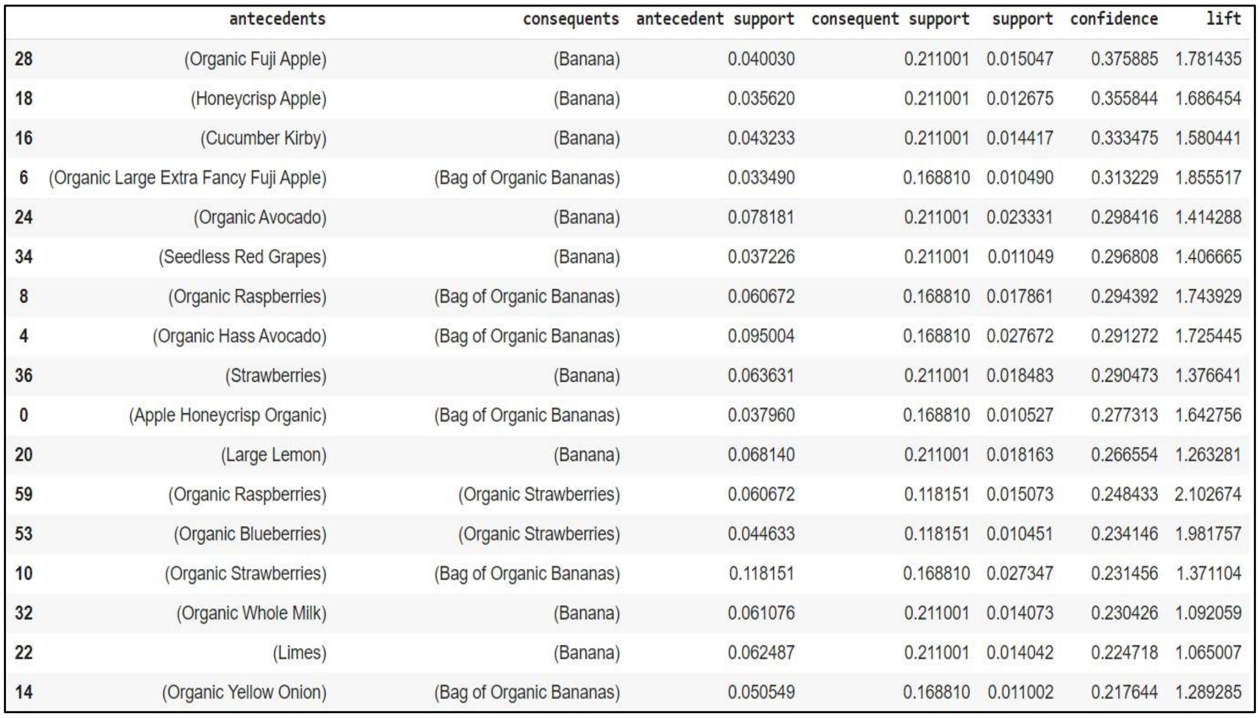


```
[12] data = products.groupby(['order_id','product_name']).size().reset_index(name='count')
     ap = (data.groupby(['order_id', 'product_name'])['count'].sum().unstack().reset_index().fillna(0).set_index('order_id'))

     #The encoding function
     def encoder(i):
         if i <= 0:
             return 0
         if i >= 1:
             return 1
     set = ap.applymap(encoder)
```

Now let us apply the apriori Algorithm to our products. We will generate frequent itemsets with minimum support of at least 1% as this a more favorable support level that could show us more results.

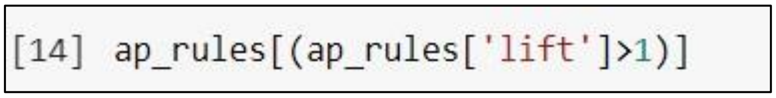


```
[13] frequent_items = apriori(set, min_support=0.01, use_colnames=True)
     ap_rules = association_rules(frequent_items, metric="lift")
     ap_rules.sort_values('confidence', ascending = False, inplace = True)
     ap_rules.drop(['leverage', 'conviction'], inplace = True, axis=1)
     ap_rules
```

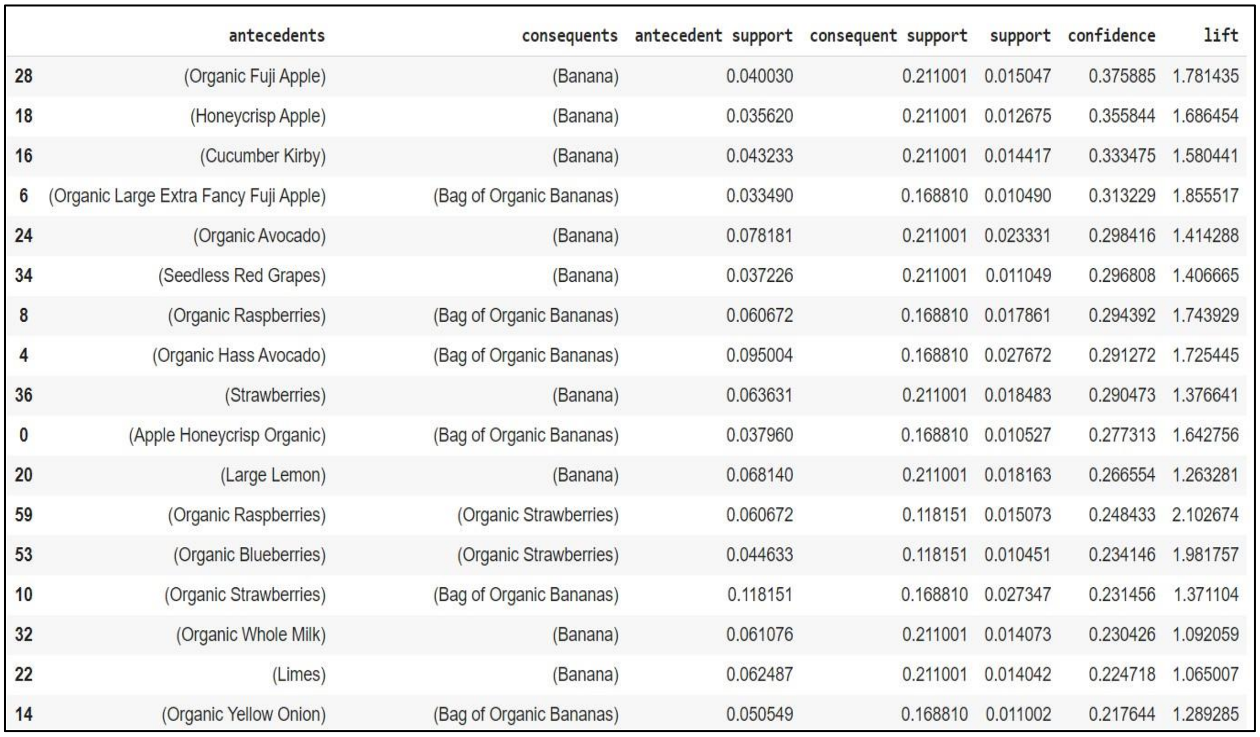


	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
28	(Organic Fuji Apple)	(Banana)	0.040030	0.211001	0.015047	0.375885	1.781435
18	(Honeycrisp Apple)	(Banana)	0.035620	0.211001	0.012675	0.355844	1.686454
16	(Cucumber Kirby)	(Banana)	0.043233	0.211001	0.014417	0.333475	1.580441
6	(Organic Large Extra Fancy Fuji Apple)	(Bag of Organic Bananas)	0.033490	0.168810	0.010490	0.313229	1.855517
24	(Organic Avocado)	(Banana)	0.078181	0.211001	0.023331	0.298416	1.414288
34	(Seedless Red Grapes)	(Banana)	0.037226	0.211001	0.011049	0.296808	1.406665
8	(Organic Raspberries)	(Bag of Organic Bananas)	0.060672	0.168810	0.017861	0.294392	1.743929
4	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.095004	0.168810	0.027672	0.291272	1.725445
36	(Strawberries)	(Banana)	0.063631	0.211001	0.018483	0.290473	1.376641
0	(Apple Honeycrisp Organic)	(Bag of Organic Bananas)	0.037960	0.168810	0.010527	0.277313	1.642756
20	(Large Lemon)	(Banana)	0.068140	0.211001	0.018163	0.266554	1.263281
59	(Organic Raspberries)	(Organic Strawberries)	0.060672	0.118151	0.015073	0.248433	2.102674
53	(Organic Blueberries)	(Organic Strawberries)	0.044633	0.118151	0.010451	0.234146	1.981757
10	(Organic Strawberries)	(Bag of Organic Bananas)	0.118151	0.168810	0.027347	0.231456	1.371104
32	(Organic Whole Milk)	(Banana)	0.061076	0.211001	0.014073	0.230426	1.092059
22	(Limes)	(Banana)	0.062487	0.211001	0.014042	0.224718	1.065007
14	(Organic Yellow Onion)	(Bag of Organic Bananas)	0.050549	0.168810	0.011002	0.217644	1.289285

After generating the association rules, we can filter out those rules based on the value of lift. We know that if lift>1, it means that there is a positive relationship between the two products. So, we will keep only those rules which have a lift greater than one.



```
[14] ap_rules[(ap_rules['lift']>1)]
```



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
28	(Organic Fuji Apple)	(Banana)	0.040030	0.211001	0.015047	0.375885	1.781435
18	(Honeycrisp Apple)	(Banana)	0.035620	0.211001	0.012675	0.355844	1.686454
16	(Cucumber Kirby)	(Banana)	0.043233	0.211001	0.014417	0.333475	1.580441
6	(Organic Large Extra Fancy Fuji Apple)	(Bag of Organic Bananas)	0.033490	0.168810	0.010490	0.313229	1.855517
24	(Organic Avocado)	(Banana)	0.078181	0.211001	0.023331	0.298416	1.414288
34	(Seedless Red Grapes)	(Banana)	0.037226	0.211001	0.011049	0.296808	1.406665
8	(Organic Raspberries)	(Bag of Organic Bananas)	0.060672	0.168810	0.017861	0.294392	1.743929
4	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.095004	0.168810	0.027672	0.291272	1.725445
36	(Strawberries)	(Banana)	0.063631	0.211001	0.018483	0.290473	1.376641
0	(Apple Honeycrisp Organic)	(Bag of Organic Bananas)	0.037960	0.168810	0.010527	0.277313	1.642756
20	(Large Lemon)	(Banana)	0.068140	0.211001	0.018163	0.266554	1.263281
59	(Organic Raspberries)	(Organic Strawberries)	0.060672	0.118151	0.015073	0.248433	2.102674
53	(Organic Blueberries)	(Organic Strawberries)	0.044633	0.118151	0.010451	0.234146	1.981757
10	(Organic Strawberries)	(Bag of Organic Bananas)	0.118151	0.168810	0.027347	0.231456	1.371104
32	(Organic Whole Milk)	(Banana)	0.061076	0.211001	0.014073	0.230426	1.092059
22	(Limes)	(Banana)	0.062487	0.211001	0.014042	0.224718	1.065007
14	(Organic Yellow Onion)	(Bag of Organic Bananas)	0.050549	0.168810	0.011002	0.217644	1.289285

From the above association rules, we can see that based on the previous data, if the user is ordering Organic Fuji Apple, then he is more likely to order bananas as well. If he is ordering Honeycrisp apples, then he is more likely to order bananas. If he is ordering Organic Raspberries, then he is more likely to add Organic Strawberries.

From the output of the above result, we can observe that the top associations are not surprising. Now, one common application of association rules mining is in the domain of recommender systems. Once item pairs have been identified as having a positive relationship, recommendations can be made to customers in order to increase sales.

7. Recommendation

By definition, A recommendation system is a system that identifies and provides recommended content or digital items for users by using users' interests. Recommender systems have become an important feature in modern websites, e.g., Amazon, Netflix, or Flickr. Click rates, revenues, and other measures of success may be increased by the application of effective recommender systems.

The difficult task is to identify relevant items even if they are generally unpopular and then predict the relevant items to the users.

Recommender systems are of various types, but in this project, we will be making use of the collaborative filtering recommender system. The collaborative filtering recommender system is a method of making automatic predictions about the transactional data of the users. Collaborative filtering is an efficient way of recommending items to users based on user history.

So, in our project, we will be implementing a collaborative filtering recommender system to predict the next items for the users based on the current items that are known positives. For this, we will be using the LightFm library available in python.

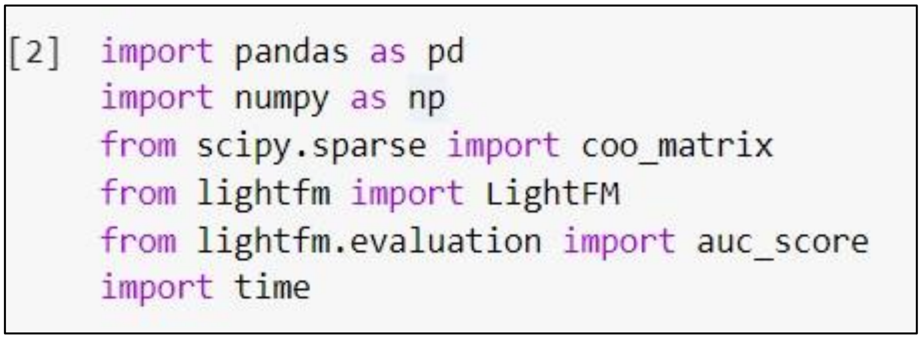
LightFM is a hybrid matrix factorization model representing users and items as linear combinations of their content features' latent factors. The biggest benefit of lightfm library is that it implements WARP (Weighted Approximate-Rank Pairwise) loss for implicit feedback learning-to-rank.

Following are the steps involved in implementing the Recommender system using LightFm:

1. Process our Data and Make a lightFM dataset by using its api
2. Build interaction matrix, user/item features
3. Make a model and train the model
4. Evaluate the model
5. Make predictions

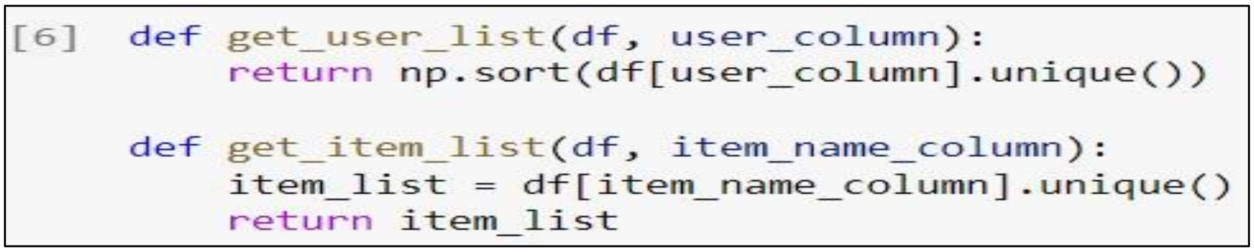
Let us look at the python code used for our Recommendation System.

First, we will import all the required libraries.

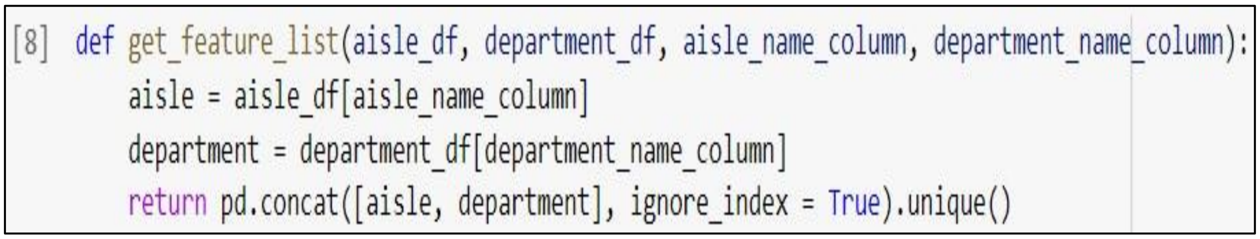


```
[2]  import pandas as pd
     import numpy as np
     from scipy.sparse import coo_matrix
     from lightfm import LightFM
     from lightfm.evaluation import auc_score
     import time
```

Now, let us create a list of users and a list of items from our dataframe and; user_column is a column consisting of users in our data frame and, item_column is a column consisting of items in our dataframe.

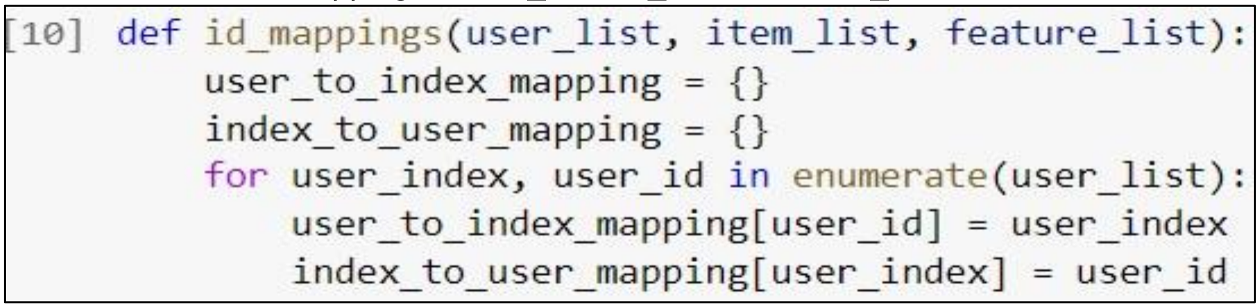


```
[6]  def get_user_list(df, user_column):
         return np.sort(df[user_column].unique())

     def get_item_list(df, item_name_column):
         item_list = df[item_name_column].unique()
         return item_list
```



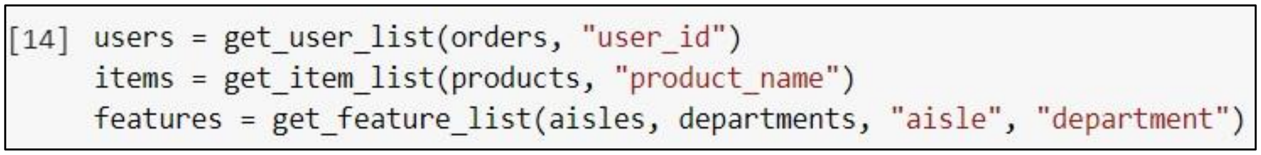
```
[8]  def get_feature_list(aisle_df, department_df, aisle_name_column, department_name_column):
         aisle = aisle_df[aisle_name_column]
         department = department_df[department_name_column]
         return pd.concat([aisle, department], ignore_index = True).unique()
```

Then, let's create id mapping for user_id, item_id, and feature_id.



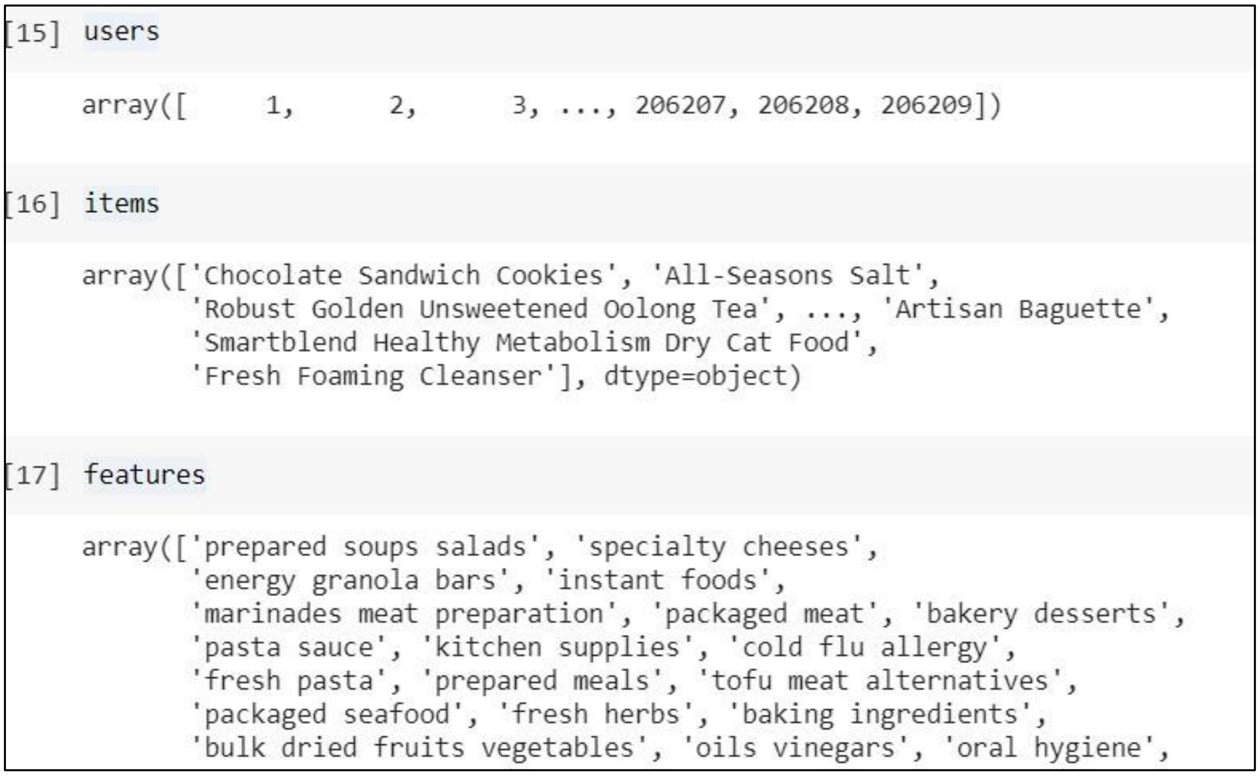
```
[10] def id_mappings(user_list, item_list, feature_list):
         user_to_index_mapping = {}
         index_to_user_mapping = {}
         for user_index, user_id in enumerate(user_list):
             user_to_index_mapping[user_id] = user_index
             index_to_user_mapping[user_index] = user_id
```

Let's create the user, item, and features list.

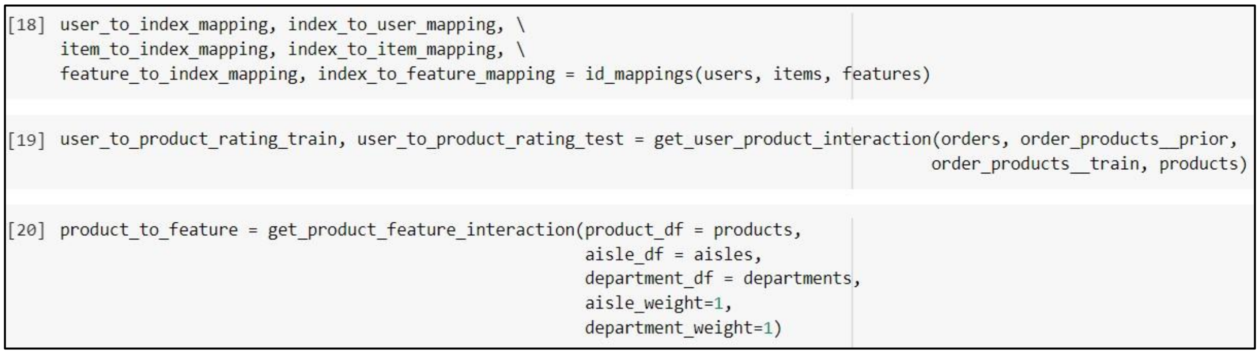


```
[14] users = get_user_list(orders, "user_id")
     items = get_item_list(products, "product_name")
     features = get_feature_list(aisles, departments, "aisle", "department")
```

Following is the list of users, items, and features.



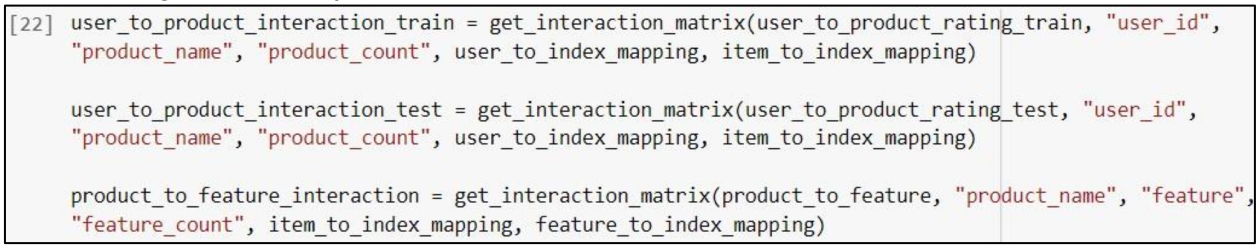
```
[15] users

     array([     1,      2,      3, ..., 206207, 206208, 206209])

[16] items

     array(['Chocolate Sandwich Cookies', 'All-Seasons Salt',
            'Robust Golden Unsweetened Oolong Tea', ..., 'Artisan Baguette',
            'Smartblend Healthy Metabolism Dry Cat Food',
            'Fresh Foaming Cleanser'], dtype=object)

[17] features

     array(['prepared soups salads', 'specialty cheeses',
            'energy granola bars', 'instant foods',
            'marinades meat preparation', 'packaged meat', 'bakery desserts',
            'pasta sauce', 'kitchen supplies', 'cold flu allergy',
            'fresh pasta', 'prepared meals', 'tofu meat alternatives',
            'packaged seafood', 'fresh herbs', 'baking ingredients',
            'bulk dried fruits vegetables', 'oils vinegars', 'oral hygiene',
```



```
[18] user_to_index_mapping, index_to_user_mapping, \
     item_to_index_mapping, index_to_item_mapping, \
     feature_to_index_mapping, index_to_feature_mapping = id_mappings(users, items, features)

[19] user_to_product_rating_train, user_to_product_rating_test = get_user_product_interaction(orders, order_products__prior,
                                                                                                 order_products__train, products)

[20] product_to_feature = get_product_feature_interaction(product_df = products,
                                                          aisle_df = aisles,
                                                          department_df = departments,
                                                          aisle_weight=1,
                                                          department_weight=1)
```

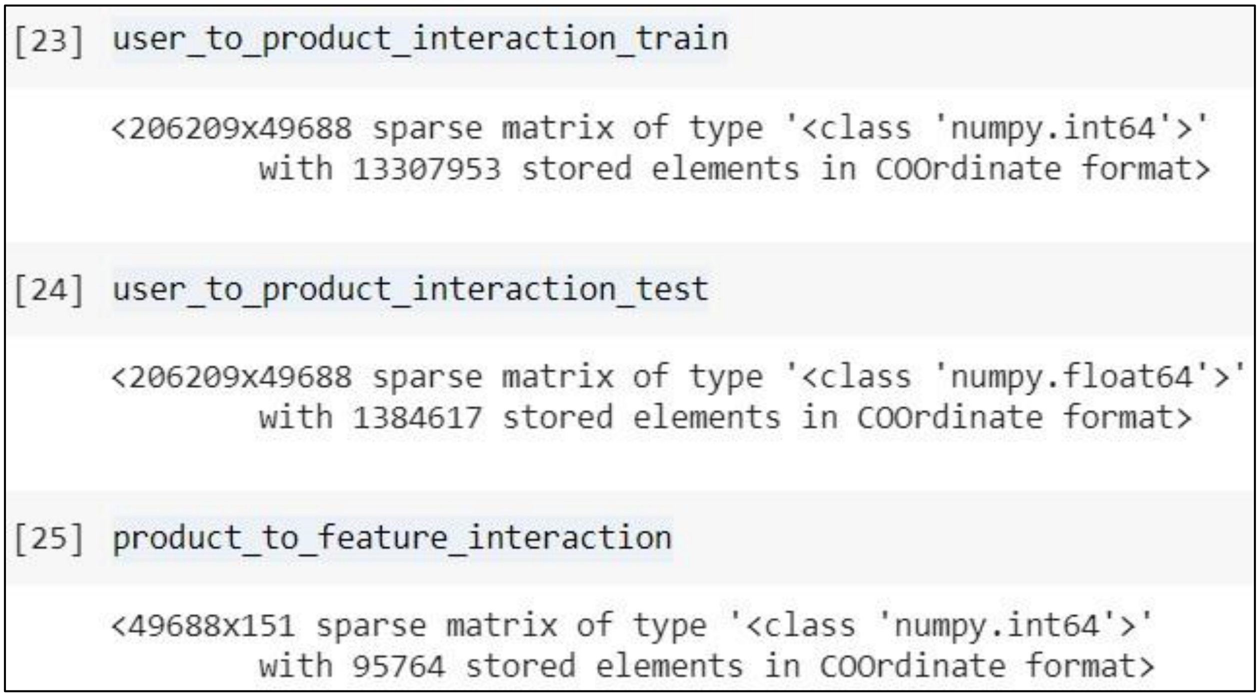
Now, to create a recommendation system using LightFm, we need to create an interaction matrix first.

Then, let's generate a sparse matrix for train and test data.



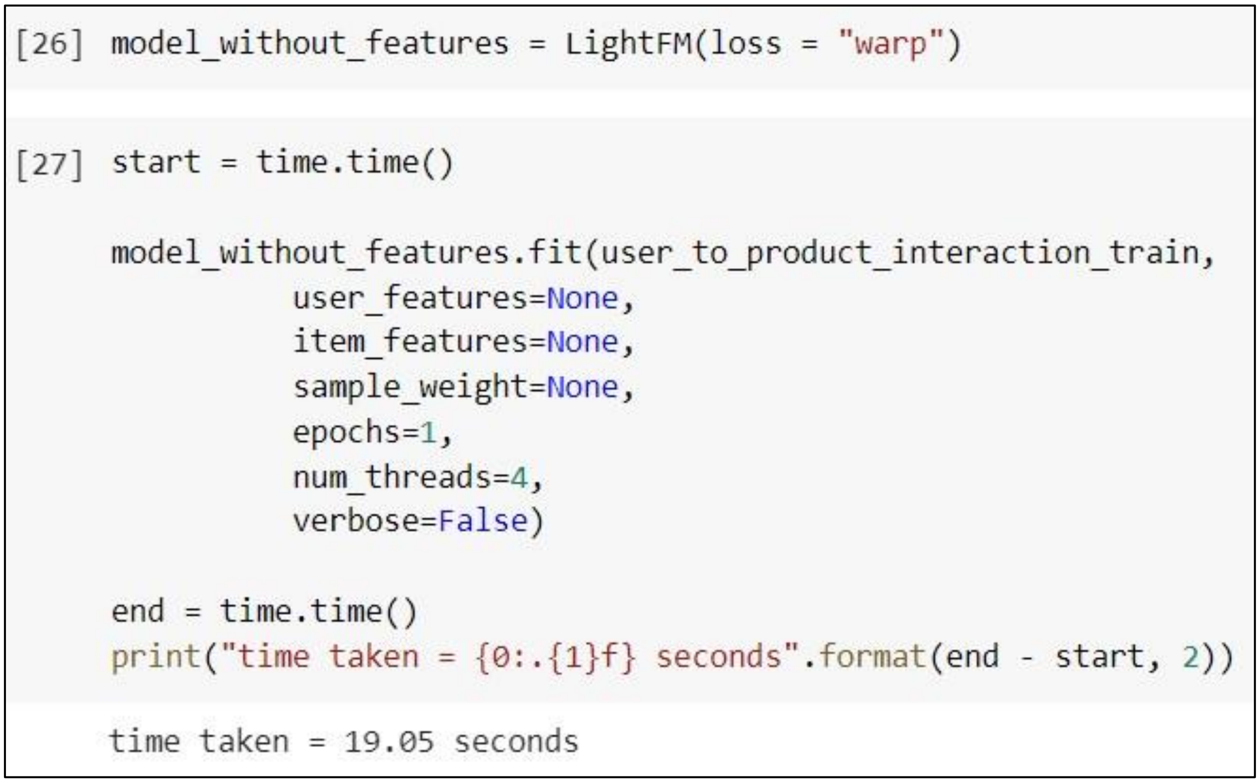
```
[22] user_to_product_interaction_train = get_interaction_matrix(user_to_product_rating_train, "user_id",
     "product_name", "product_count", user_to_index_mapping, item_to_index_mapping)

     user_to_product_interaction_test = get_interaction_matrix(user_to_product_rating_test, "user_id",
     "product_name", "product_count", user_to_index_mapping, item_to_index_mapping)

     product_to_feature_interaction = get_interaction_matrix(product_to_feature, "product_name", "feature",
     "feature_count", item_to_index_mapping, feature_to_index_mapping)
```

Following are the sparse matrices that are generated.

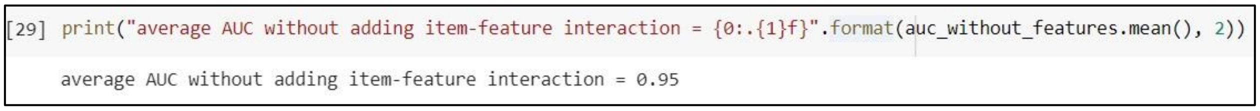


```
[23] user_to_product_interaction_train

     <206209x49688 sparse matrix of type '<class 'numpy.int64'>'
             with 13307953 stored elements in COOrdinate format>

[24] user_to_product_interaction_test

     <206209x49688 sparse matrix of type '<class 'numpy.float64'>'
             with 1384617 stored elements in COOrdinate format>

[25] product_to_feature_interaction

     <49688x151 sparse matrix of type '<class 'numpy.int64'>'
             with 95764 stored elements in COOrdinate format>
```

Then we will apply LightFM cross-validation using collaborative filtering and the warp loss for implicit learning feedback.



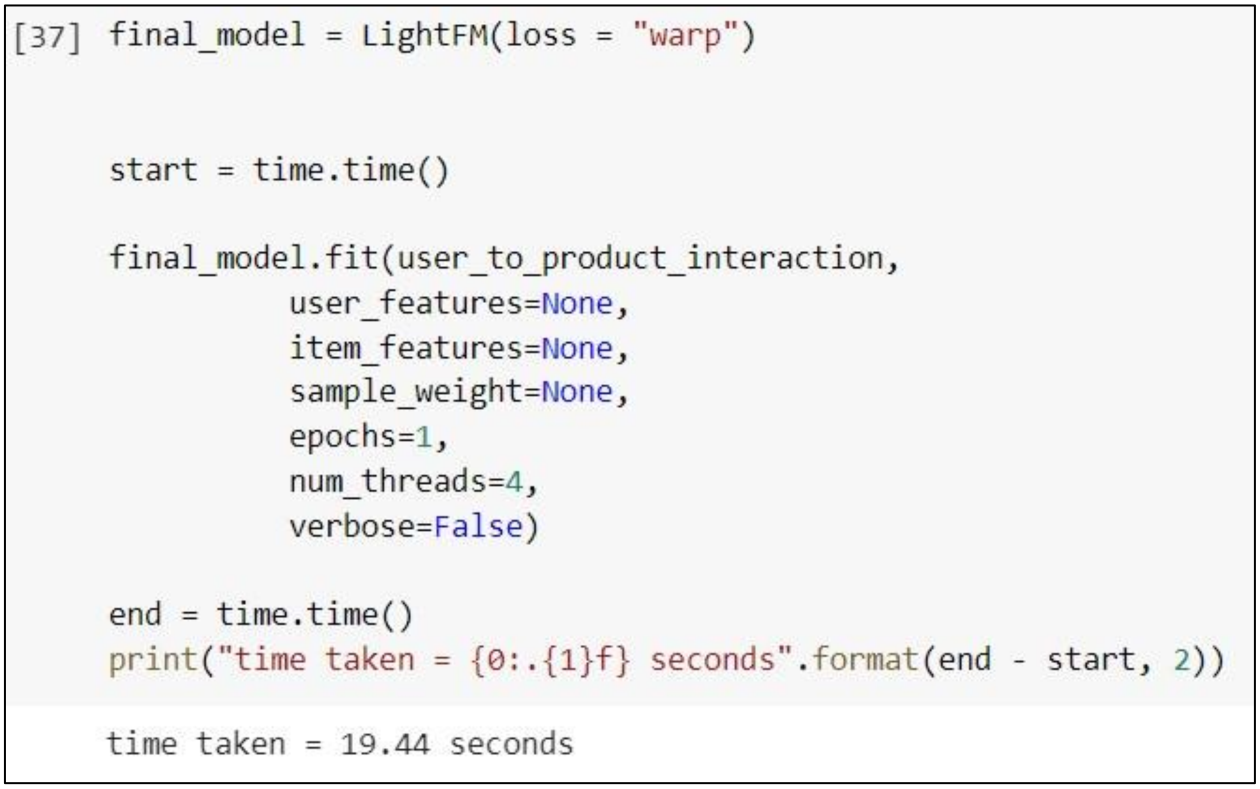
```
[26] model_without_features = LightFM(loss = "warp")

[27] start = time.time()

     model_without_features.fit(user_to_product_interaction_train,
                user_features=None,
                item_features=None,
                sample_weight=None,
                epochs=1,
                num_threads=4,
                verbose=False)

     end = time.time()
     print("time taken = {0:.{1}f} seconds".format(end - start, 2))

     time taken = 19.05 seconds
```



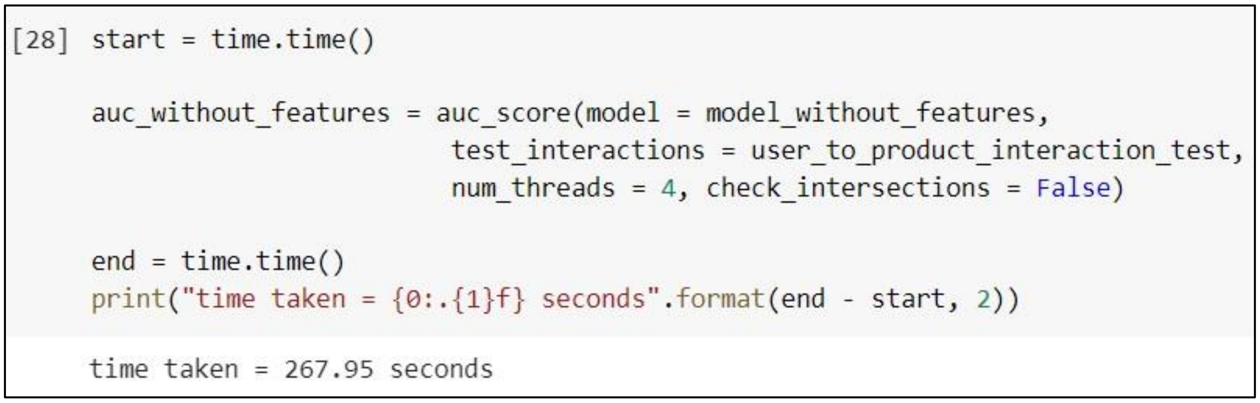
```
[29] print("average AUC without adding item-feature interaction = {0:.{1}f}".format(auc_without_features.mean(), 2))

     average AUC without adding item-feature interaction = 0.95
```

Also, we will calculate the AUC need for this Algorithm, which is 0.95 Let

us retain the final model.

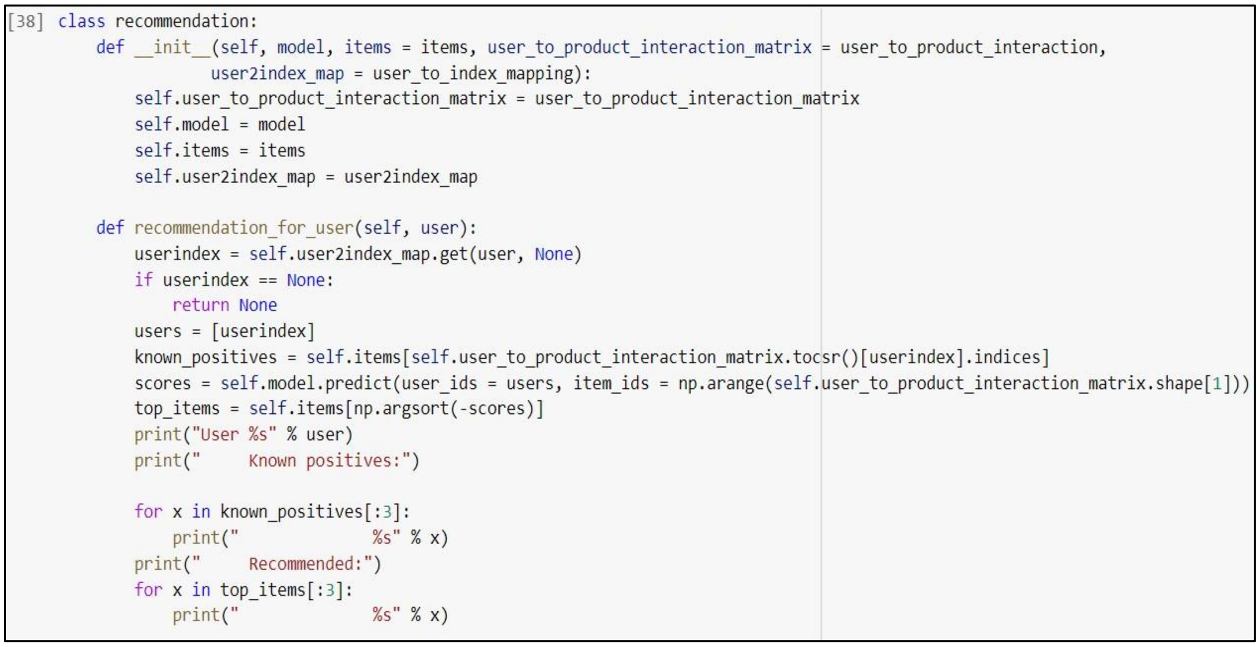


```
[37] final_model = LightFM(loss = "warp")


     start = time.time()

     final_model.fit(user_to_product_interaction,
                     user_features=None,
                     item_features=None,
                     sample_weight=None,
                     epochs=1,
                     num_threads=4,
                     verbose=False)

     end = time.time()
     print("time taken = {0:.{1}f} seconds".format(end - start, 2))

     time taken = 19.44 seconds
```

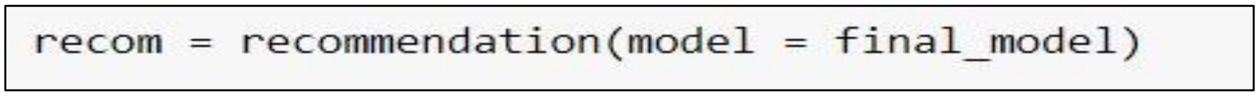


```
[28] start = time.time()

     auc_without_features = auc_score(model = model_without_features,
                                      test_interactions = user_to_product_interaction_test,
                                      num_threads = 4, check_intersections = False)

     end = time.time()
     print("time taken = {0:.{1}f} seconds".format(end - start, 2))

     time taken = 267.95 seconds
```

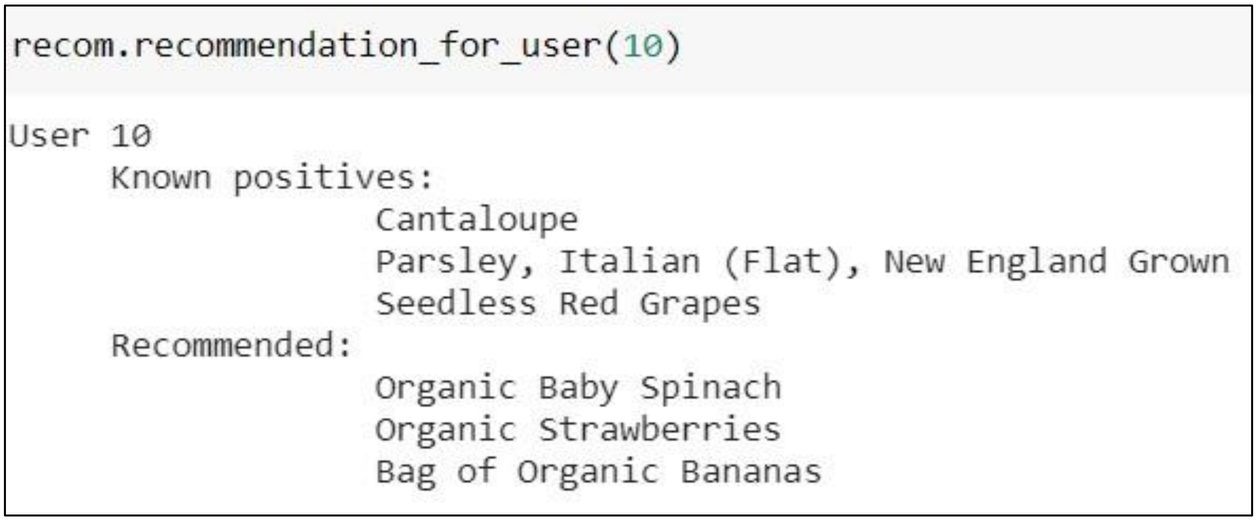
We can then use the recommendation_for_user function for generating the output. This function takes matrix factorization model, interaction matrix, user dictionary, item dictionary, user_id, and the number of items as input and returns the list of item id's a user may be interested in interacting with.


```
[38] class recommendation:
         def __init__(self, model, items = items, user_to_product_interaction_matrix = user_to_product_interaction,
                     user2index_map = user_to_index_mapping):
             self.user_to_product_interaction_matrix = user_to_product_interaction_matrix
             self.model = model
             self.items = items
             self.user2index_map = user2index_map

         def recommendation_for_user(self, user):
             userindex = self.user2index_map.get(user, None)
             if userindex == None:
                 return None
             users = [userindex]
             known_positives = self.items[self.user_to_product_interaction_matrix.tocsr()[userindex].indices]
             scores = self.model.predict(user_ids = users, item_ids = np.arange(self.user_to_product_interaction_matrix.shape[1]))
             top_items = self.items[np.argsort(-scores)]
             print("User %s" % user)
             print("     Known positives:")

             for x in known_positives[:3]:
                 print("              %s" % x)
             print("     Recommended:")
             for x in top_items[:3]:
                 print("              %s" % x)
```

We will then use the final_model to pass as a parameter to our recommendation class.


```
recom = recommendation(model = final_model)
```

And now, we will generate recommendations for the users in our dataset.



```
recom.recommendation_for_user(10)

User 10
     Known positives:
                    Cantaloupe
                    Parsley, Italian (Flat), New England Grown
                    Seedless Red Grapes
     Recommended:
                    Organic Baby Spinach
                    Organic Strawberries
                    Bag of Organic Bananas
```

From the above output, we can see that if Cantaloupe, Parsley, Seedless Red Grapes are our known positives, then Organic Baby Spinach, Organic Strawberries, and Bag of Organic Bananas will be the recommended products.

8. Conclusion

At the start of this project, we plan to provide a recommendation system for products from customers at the beginning of this project and to know their purchase habits. Market Basket Analysis is one such important methodology used to reveal the relationship rules between the goods by most major retail firms. It works by searching for the combinations that often occur in transactions in the products that occur together. To put it in simple terms, for the consumers and the dealers and the shop owners who get a positive review for the prompt delivery, this approach is profitable.

9. Future scope

Developing this project further, we can try to make a personalized shopping cart for the users based on the previous data.

Also, special offers can be designed specifically during the holiday season.

10. References

[1] https://www.kaggle.com/c/instacart-market-basket-analysis

[2] https://towardsdatascience.com/a-gentle-introduction-on-market-basket-analysisassociation-rules-fa4b986a40ce

[3] https://smartbridge.com/market-basket-analysis-101/

[4] https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithmdata-mining-r-implement

[5] https://medium.com/kaggle-blog/instacart-market-basket-analysis-feda2700cded

[6] https://towardsdatascience.com/solving-business-usecases-by-recommendersystem-using-lightfm-4ba7b3ac8e62

[7] https://www.kaggle.com/niyamatalmass/lightfm-hybrid-recommendation-system

[8] https://select-statistics.co.uk/blog/market-basket-analysis-understanding-customerbehaviour/

[9] https://pbpython.com/market-basket-analysis.html

[10] https://www.academia.edu/Documents/in/Market_Basket_Analysis

Instacart Market Basket Analysis

MITA Capstone Project

Contents

Introduction

- Market basket analysis is one such technique used by many retails to understand the purchasing patterns of the customers.
- In this project, we will be using the Instacart data set to perform market basket analysis, so that we can uncover hidden trends and patterns in our data.
- Instacart, a grocery ordering and delivery app, aims to make it easy to fill your refrigerator and pantry with your personal favorites and staples when you need them.
- The objective of this capstone is to address the following research questions:
- Which products are strongly associated with each other?
- Which products can we recommend to the users based on the past transaction?

About the Dataset

- The dataset for this project is a relational set of files describing all the customer's orders over time. The data used for this project is from the 2017 competition that Instacart hosted on Kaggle.com. The data contains a sample of over **3 million grocery orders** from more than **200,000 Instacart users**.

- Files used in the project:

- Orders.csv: This dataset has 3.4 million rows and 206K users and describes the details about each order.

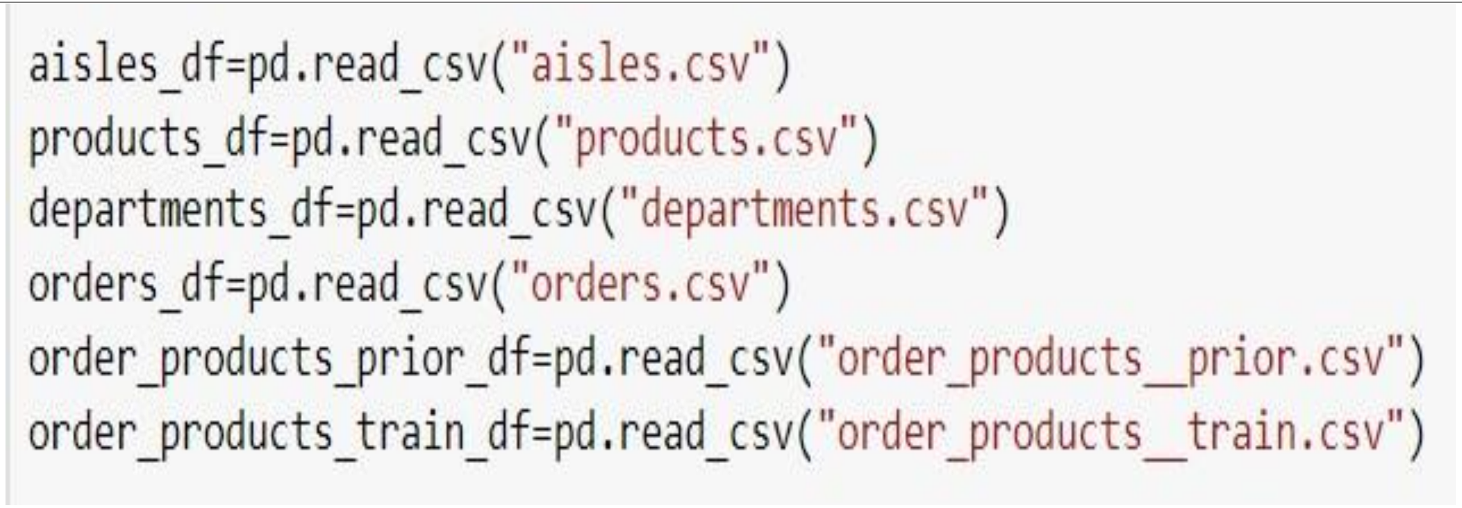
- Products.csv: This dataset has 50K rows and describes the product names, aisles and departments associated with the products.

- Aisles.csv: This dataset has 134 rows and describes the aisles of Instacart for the products.

- Departments.csv: This dataset has 21 rows and describes the departments of Instacart for the products.

- Orders_product SET: This dataset contains 30m+ records and is divided into two datasets. The orders_train_prior set and the order test prior set. The attributes for both the datasets are same.

Read the Dataset

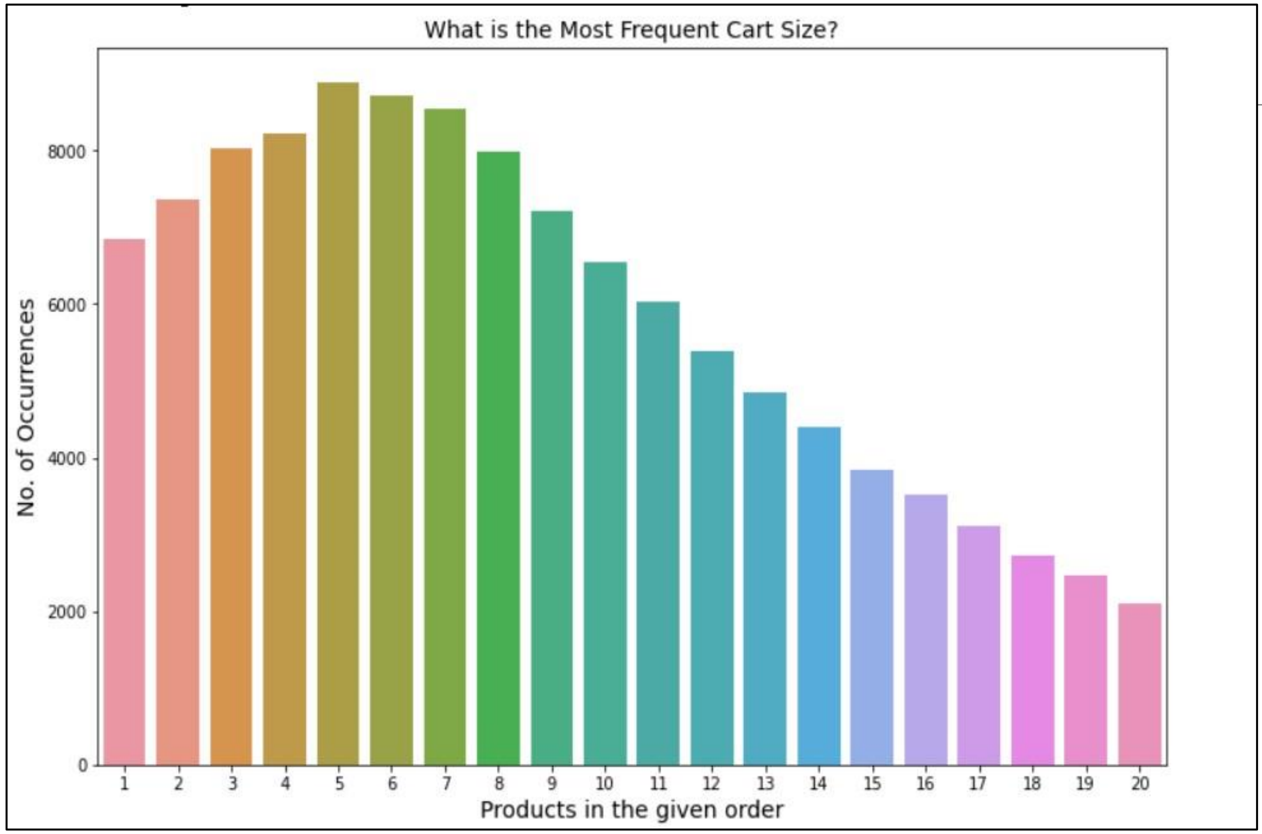


```
aisles_df=pd.read_csv("aisles.csv")
products_df=pd.read_csv("products.csv")
departments_df=pd.read_csv("departments.csv")
orders_df=pd.read_csv("orders.csv")
order_products_prior_df=pd.read_csv("order_products__prior.csv")
order_products_train_df=pd.read_csv("order_products__train.csv")
```

Exploratory Data Analysis

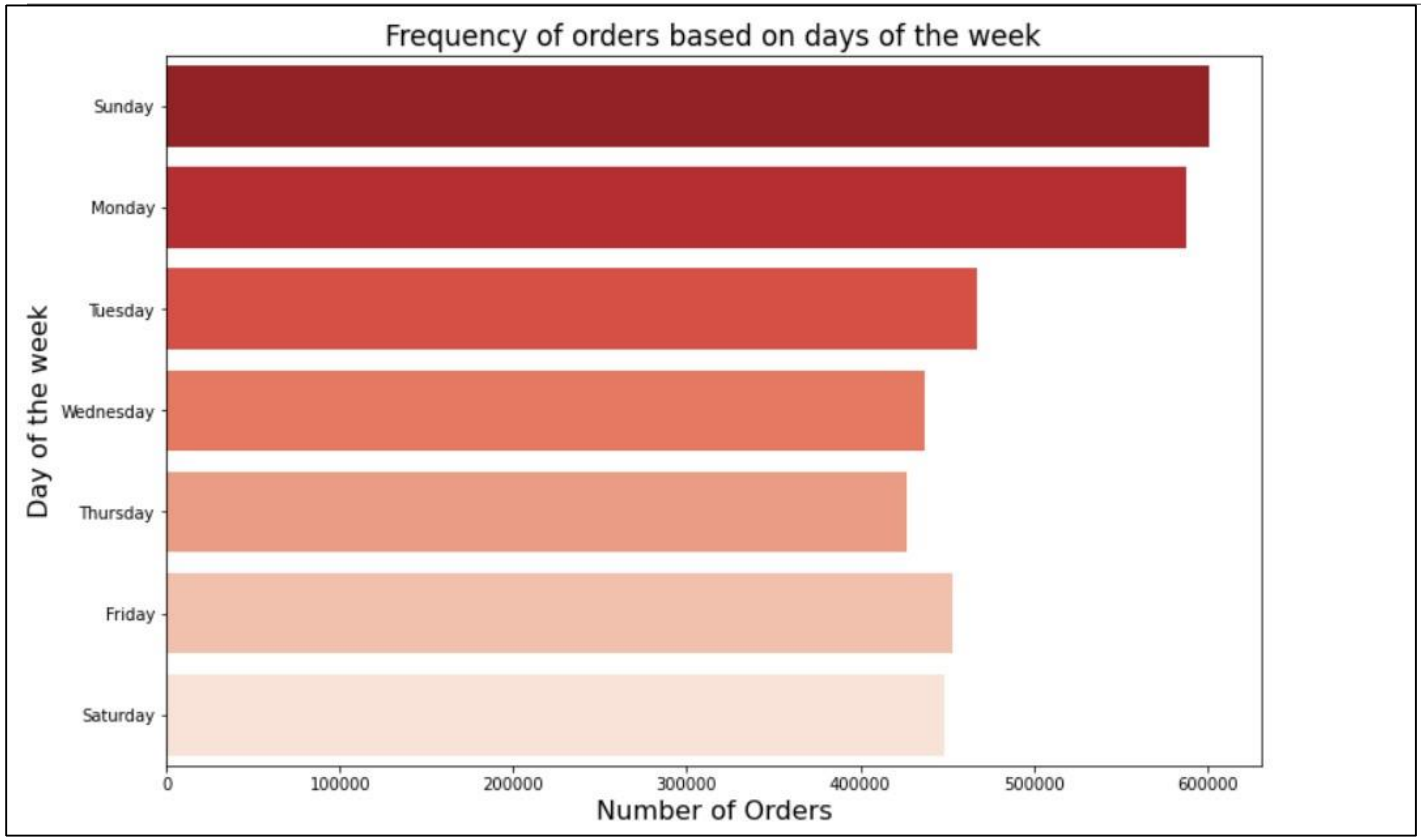
Let us have a look at few of the visualizations we performed using exploratory data analysis.

Exploratory Data Analysis



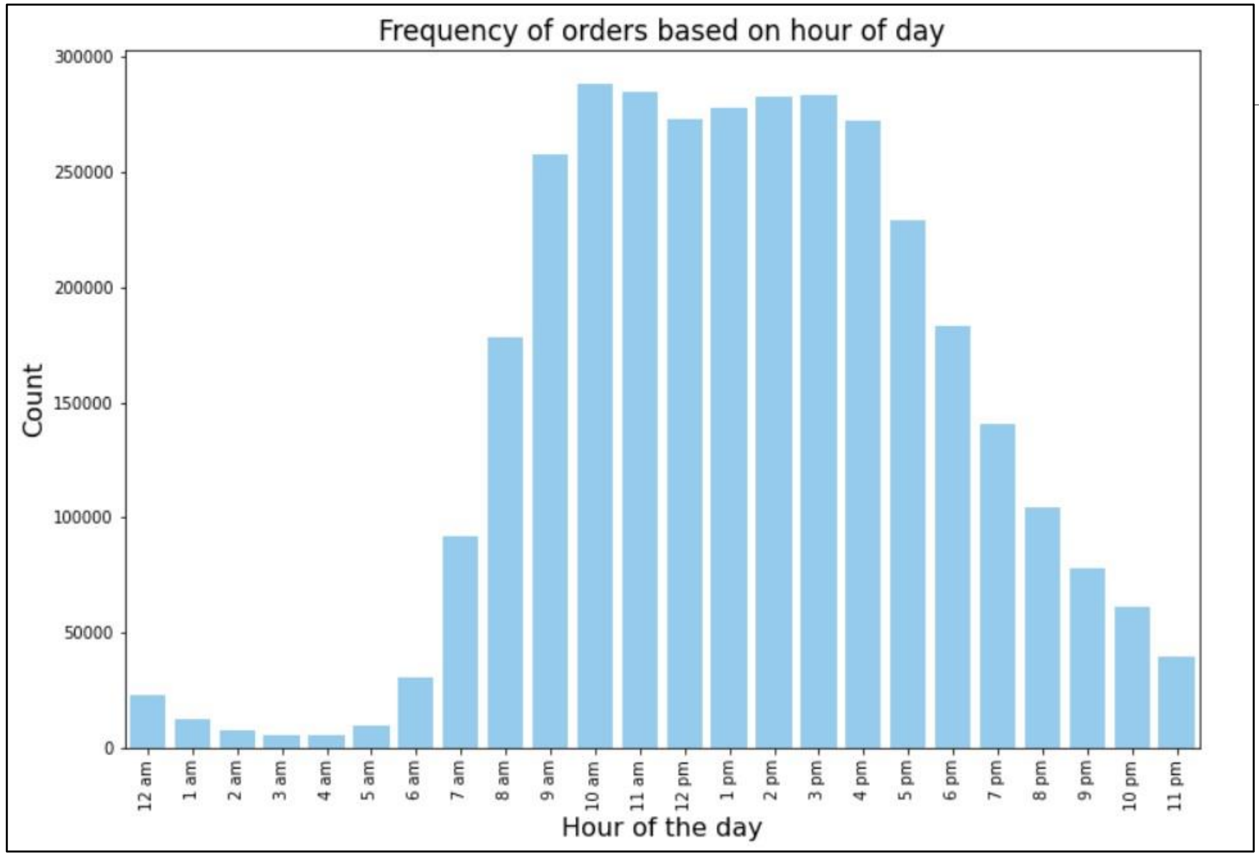
From the plot, we can see that mostly 5 items or products are ordered per order. Also, we can see peaks at 6 and 7. So on an average, 5-7 products is the most frequent cart size.

Exploratory Data Analysis



From this graph we can see that the number of orders is more on Sundays and Mondays, so it seems that people shop more either at the end of the week or at the start of the week.

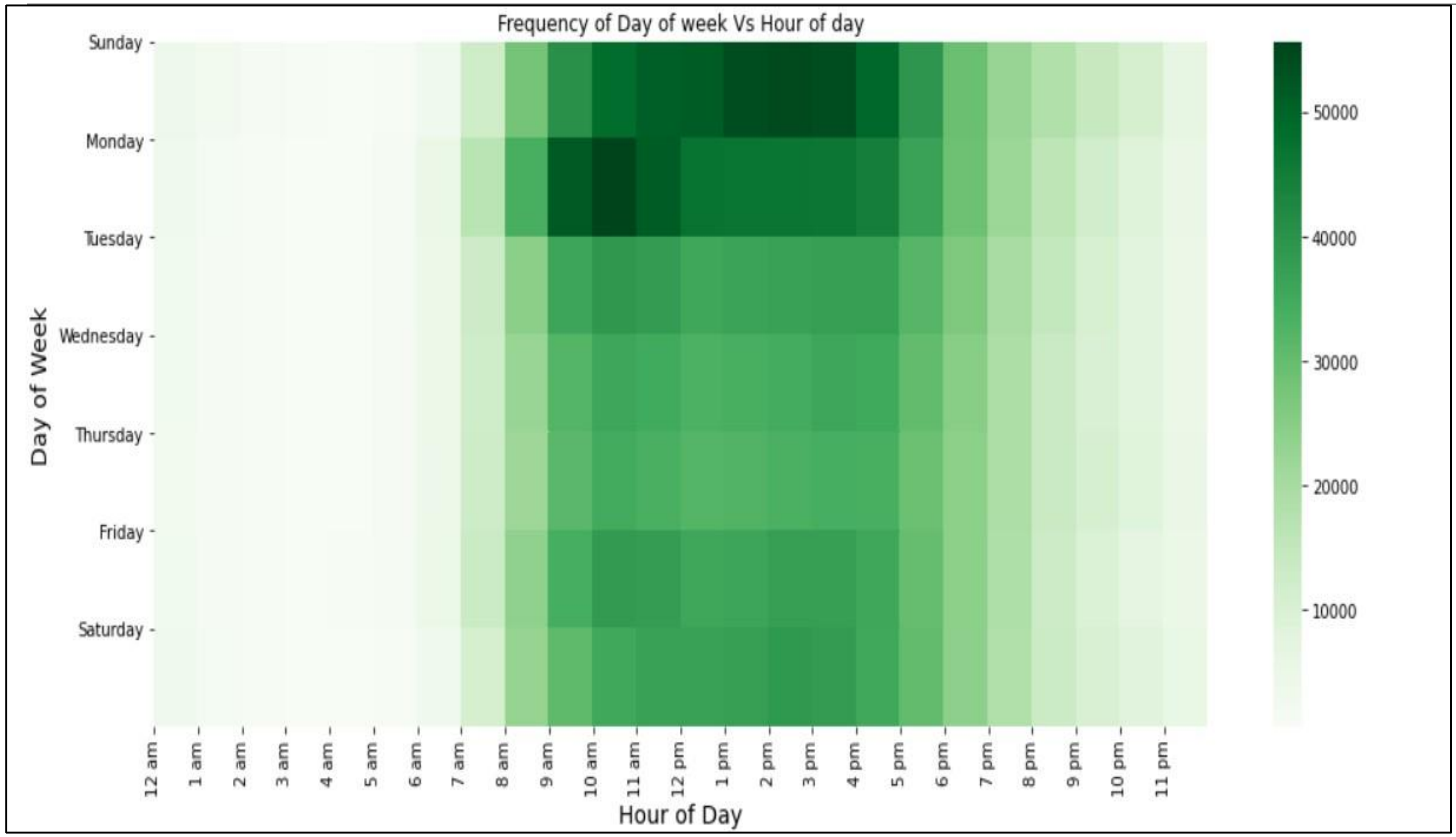
Exploratory Data Analysis



From the plot, we can clearly see that

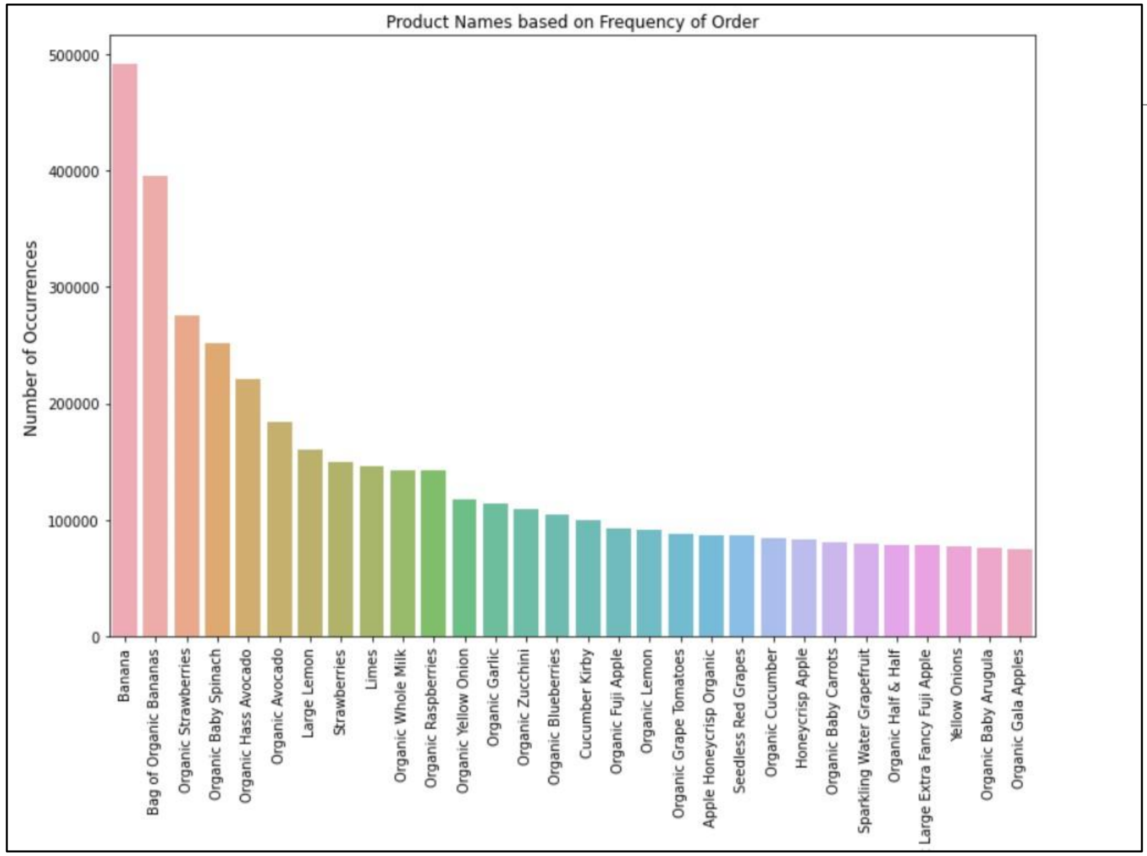
the maximum number of orders were placed at 10 am and 11 am which makes sense because that is the time when people usually start their day

Exploratory Data Analysis



From this heatmap, we can see that the peak orders are in the afternoon on Sunday and on Monday between 9 am to 4 pm. So its looks like Sundays and Mondays are the prime days for ordering.

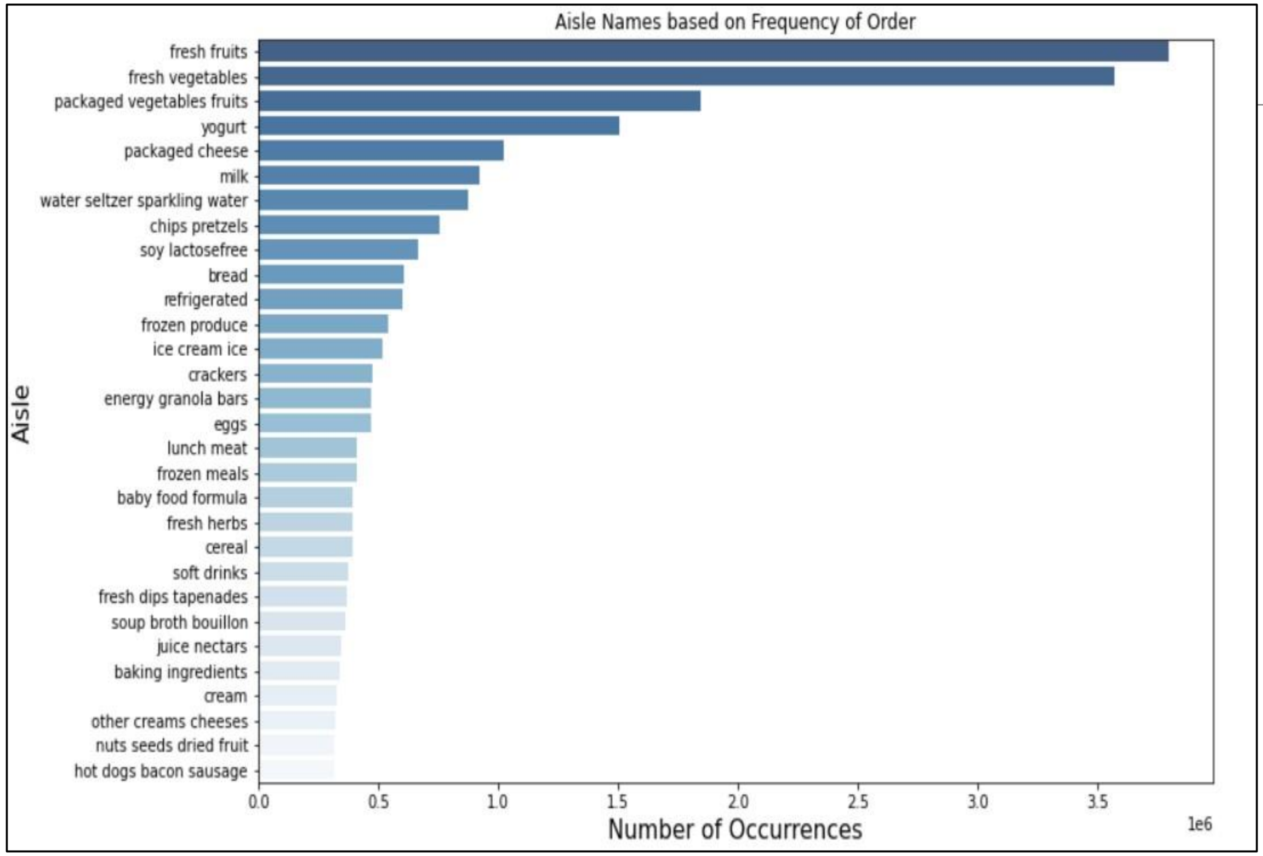
Exploratory Data Analysis



Here, we can see that Bananas and Organic

Bananas are the most frequently ordered products at Instacart. Followed by Organic Strawberries, Organic Baby Spinach, Organic Hass Avocado, Large Limes, Strawberries and so on.

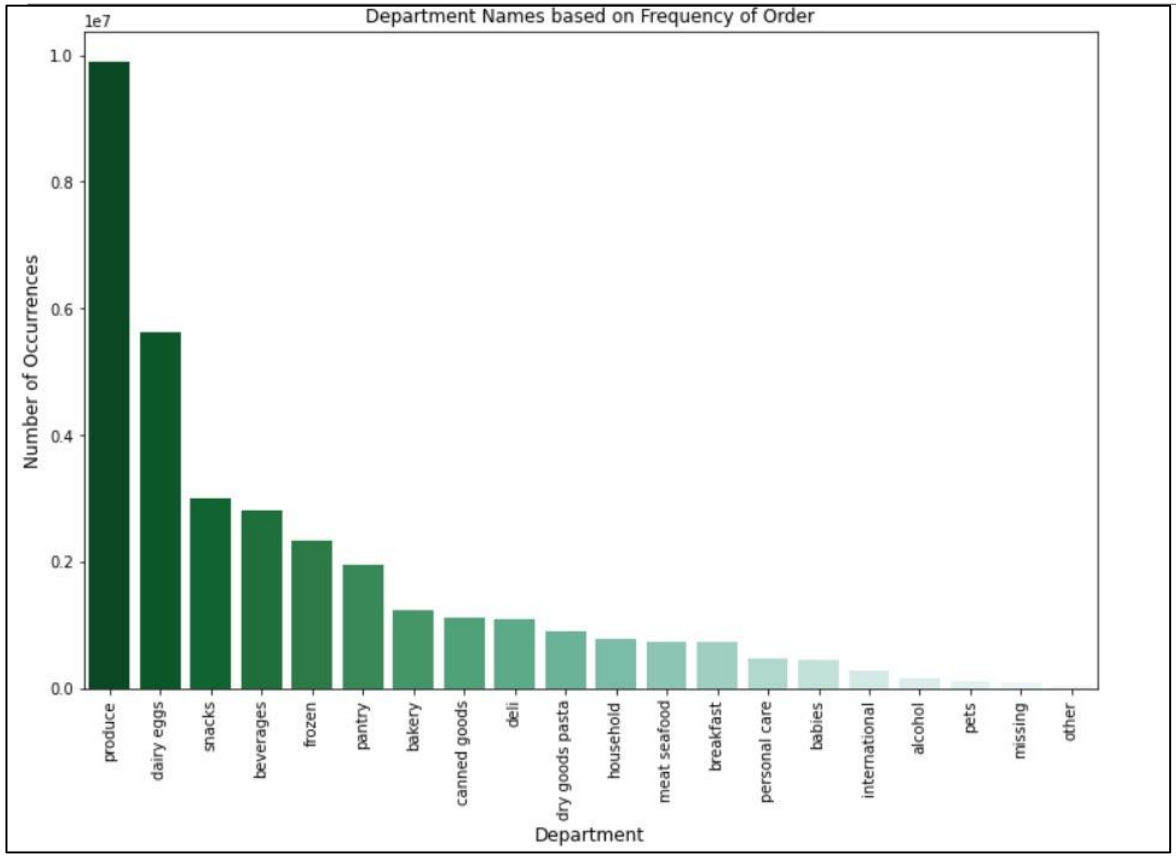
Exploratory Data Analysis



Here, we can see that fresh fruits and

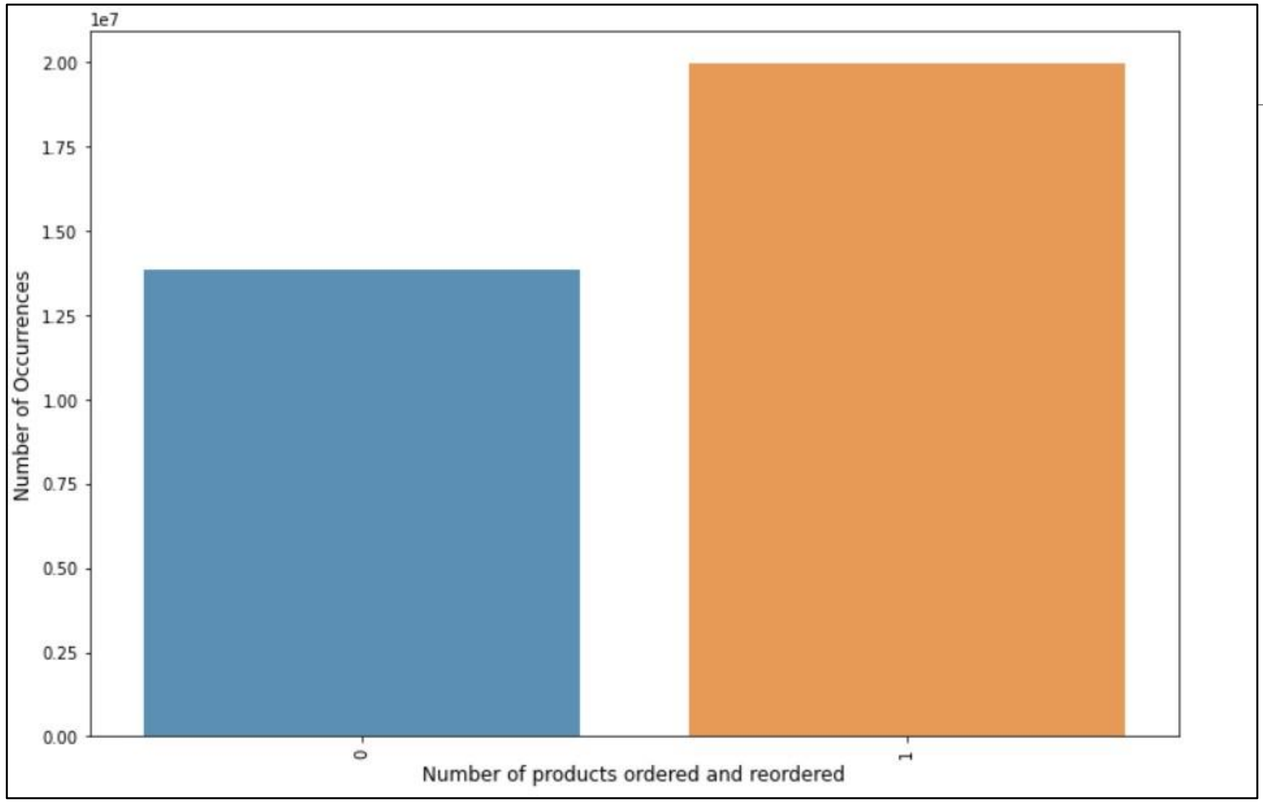
fresh vegetables are the most popular aisles at Instacart. Also, we can see demand for packaged vegetables, yogurt, packaged cheese and milk.

Exploratory Data Analysis



From this graph, we can see that the produce, dairy eggs, snacks and beverages are the most popular departments at Instacart.

Exploratory Data Analysis

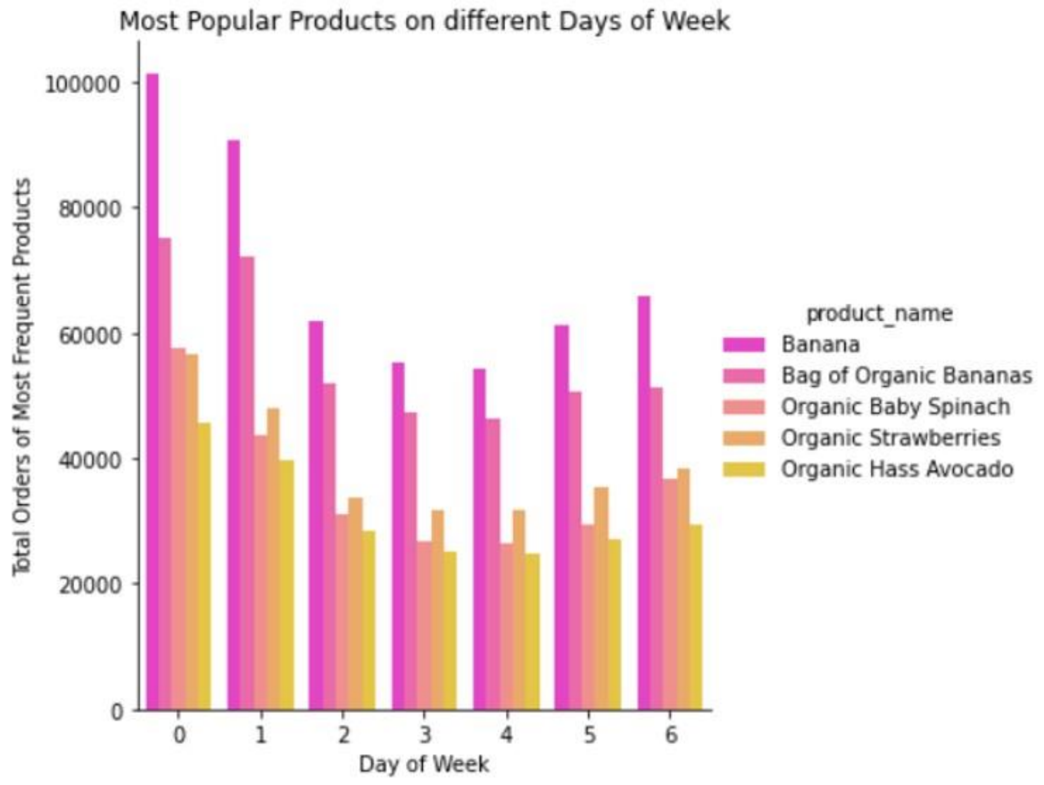


Here, we can see that the number of

occurrences of products for 1 are more. That means maximum number of products are reordered at Instacart.

Exploratory Data Analysis

Here, we can see that Bananas and Organic Bananas are the most frequent product on all days of the week.



Association Rule Mining

- Let us now have a look at how the data is actually associated with each other.
- For that we will be using the Apriori algorithm to generate association Rules.
- This will help us find out the association between different items in our dataset and how they are related.
- The Apriori algorithm is based on 3 terminologies: support,, confidence and lift.
- The confidence values range from 0 to 1 where 0 indicates that Y is never purchased when X is purchased, and 1 indicates that Y is always purchased whenever X is purchased.
- Lift = 1; implies no relationship between X and Y (i.e., X and Y occur together only by chance)

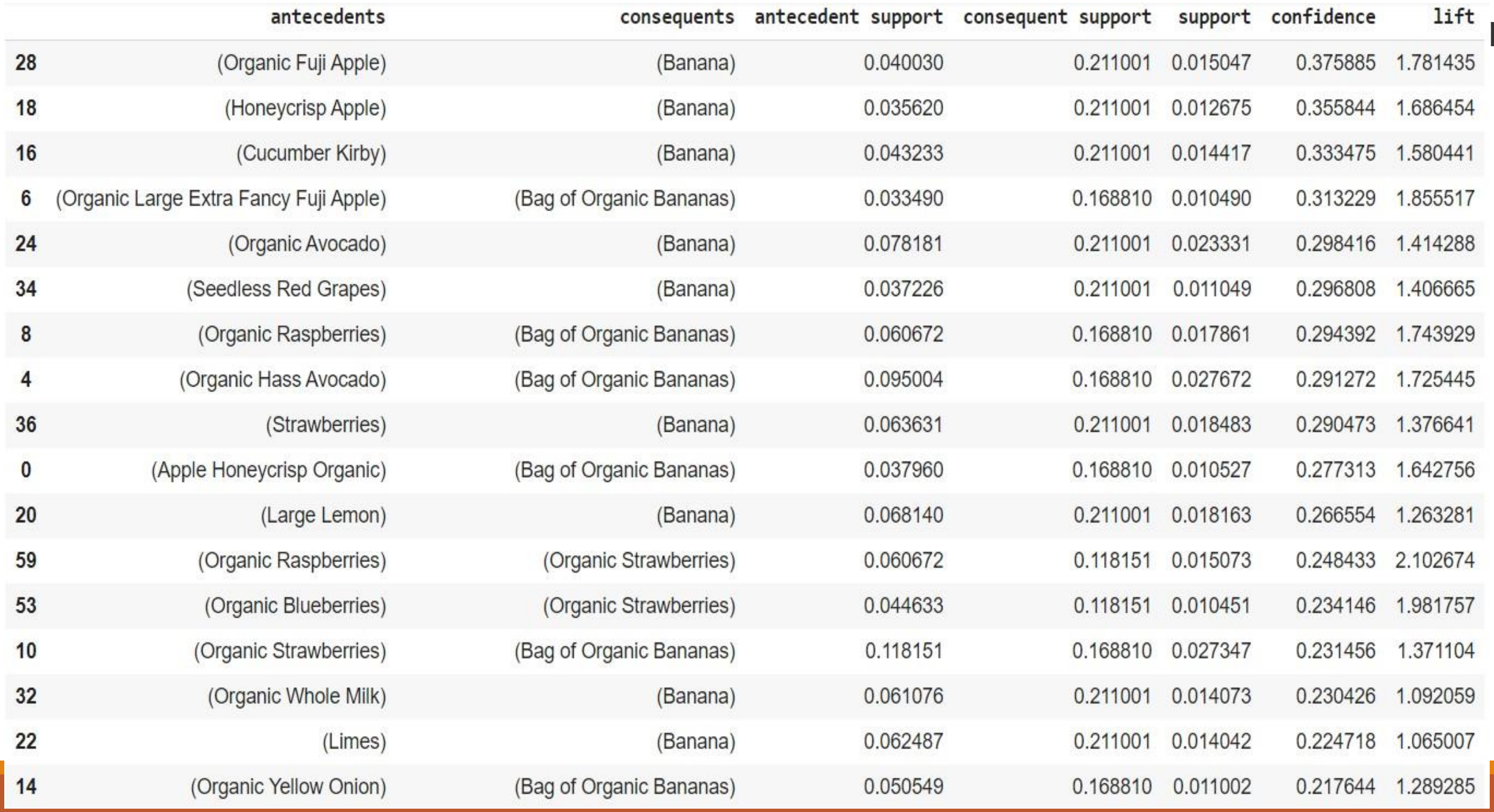
Recommendation System

- Lift > 1; implies that there is a positive relationship between X and Y (i.e., X and Y occur together more often than random)
- Lift < 1; implies that there is a negative relationship between X and Y (i.e., X and Y occur together less often than random)

Association Rule Mining

Recommendation System

ri algorithm.



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
28	(Organic Fuji Apple)	(Banana)	0.040030	0.211001	0.015047	0.375885	1.781435
18	(Honeycrisp Apple)	(Banana)	0.035620	0.211001	0.012675	0.355844	1.686454
16	(Cucumber Kirby)	(Banana)	0.043233	0.211001	0.014417	0.333475	1.580441
6	(Organic Large Extra Fancy Fuji Apple)	(Bag of Organic Bananas)	0.033490	0.168810	0.010490	0.313229	1.855517
24	(Organic Avocado)	(Banana)	0.078181	0.211001	0.023331	0.298416	1.414288
34	(Seedless Red Grapes)	(Banana)	0.037226	0.211001	0.011049	0.296808	1.406665
8	(Organic Raspberries)	(Bag of Organic Bananas)	0.060672	0.168810	0.017861	0.294392	1.743929
4	(Organic Hass Avocado)	(Bag of Organic Bananas)	0.095004	0.168810	0.027672	0.291272	1.725445
36	(Strawberries)	(Banana)	0.063631	0.211001	0.018483	0.290473	1.376641
0	(Apple Honeycrisp Organic)	(Bag of Organic Bananas)	0.037960	0.168810	0.010527	0.277313	1.642756
20	(Large Lemon)	(Banana)	0.068140	0.211001	0.018163	0.266554	1.263281
59	(Organic Raspberries)	(Organic Strawberries)	0.060672	0.118151	0.015073	0.248433	2.102674
53	(Organic Blueberries)	(Organic Strawberries)	0.044633	0.118151	0.010451	0.234146	1.981757
10	(Organic Strawberries)	(Bag of Organic Bananas)	0.118151	0.168810	0.027347	0.231456	1.371104
32	(Organic Whole Milk)	(Banana)	0.061076	0.211001	0.014073	0.230426	1.092059
22	(Limes)	(Banana)	0.062487	0.211001	0.014042	0.224718	1.065007
14	(Organic Yellow Onion)	(Bag of Organic Bananas)	0.050549	0.168810	0.011002	0.217644	1.289285

- By definition, a recommendation system is a system that identifies and provides recommended content or digital items for users by using users interests.
- Recommender systems are of various types, but in this project, we will be making use of the collaborative filtering recommender system.
- In our project we will be implementing a collaborative filtering recommender system to predict the next items for the users based on the current items that is known positives. For this, we will be using the LightFm library available in python.
- Let us look at the implementation of our recommendation system.

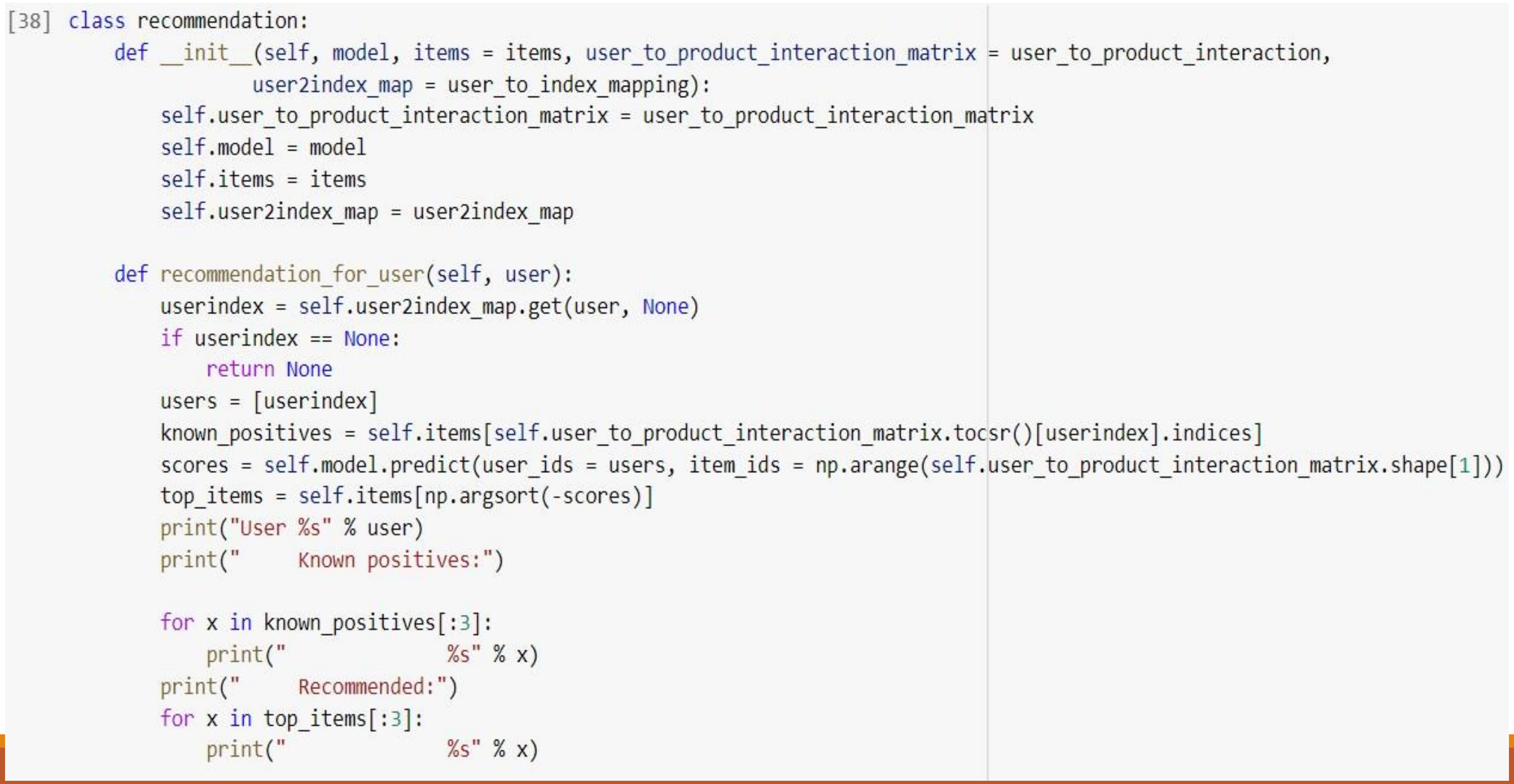
Recommendation System

- For our recommendation system we will process our data using the LightFm library first.
- Then we will build interaction matrix using the user and item features.
- Then, we will make a model and train that model

Recommendation System

- Then we will evaluate the model
- And then make the predictions.
- Let us have a look at the predictions made by our recommendation system.

Recommendation System



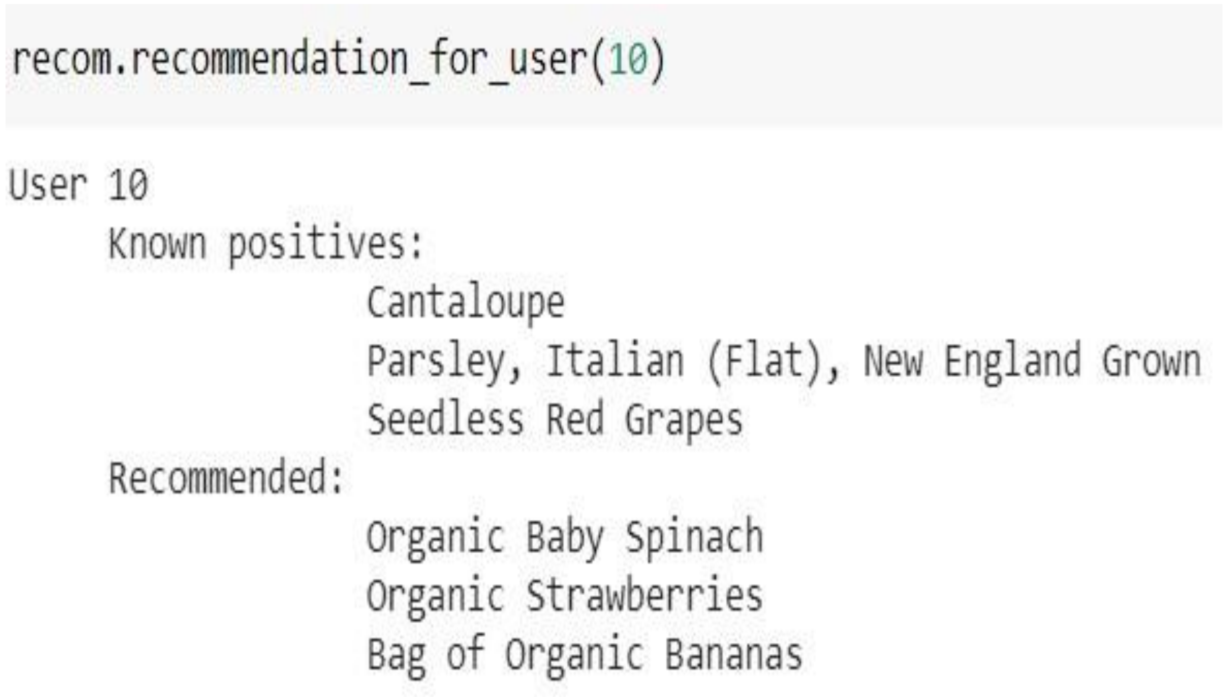
```
[38] class recommendation:
         def __init__(self, model, items = items, user_to_product_interaction_matrix = user_to_product_interaction,
                      user2index_map = user_to_index_mapping):
             self.user_to_product_interaction_matrix = user_to_product_interaction_matrix
             self.model = model
             self.items = items
             self.user2index_map = user2index_map

         def recommendation_for_user(self, user):
             userindex = self.user2index_map.get(user, None)
             if userindex == None:
                 return None
             users = [userindex]
             known_positives = self.items[self.user_to_product_interaction_matrix.tocsr()[userindex].indices]
             scores = self.model.predict(user_ids = users, item_ids = np.arange(self.user_to_product_interaction_matrix.shape[1]))
             top_items = self.items[np.argsort(-scores)]
             print("User %s" % user)
             print("     Known positives:")

             for x in known_positives[:3]:
                 print("              %s" % x)
             print("     Recommended:")
             for x in top_items[:3]:
                 print("              %s" % x)
```

This is the code for

Recommendation System

generating our recommendations.

Recommendation System



```
recom.recommendation_for_user(10)
```

```
User 10
     Known positives:
                  Cantaloupe
                  Parsley, Italian (Flat), New England Grown
                  Seedless Red Grapes
     Recommended:
                  Organic Baby Spinach
                  Organic Strawberries
                  Bag of Organic Bananas
```

Recommendation System

Canatloupe, Parsley, Seedless Red
Grapes are our known positives, then
Organic Baby Spinach, Organic
Strawberries and Bag of Organic Bananas will be the recommended products.

Conclusion

- In this project, we analyzed the ordering patterns and trends of the Instacart users to obtain some insights using exploratory data analysis.

- Then, we used association rule mining using apriori algorithm to identify association between different products at Instacart.
- And then, we build a recommendation system by training a model to recommend products to the users based on the previous transactions at Instacart.

Thank You!!!!!