Instacart Market Basket Analysis

Surya Ramadoss

BDAP3PT - SP Jain School of Global Management

Abstract— Association rule mining is the powerful tool now a days in Data mining. It identifies the correlation between the items in large databases. A typical example of Association rule mining is Market Basket analysis. In this method or approach it examines the buying habits of the customers by identifying the associations among the items purchased by the customers in their baskets. This helps to increase in the sales of a particular product by identifying the frequent items purchased by the customers. This project is to predict which products user will buy again in Instacart Market Basket data set

Keywords— Market Basket Analysis, PCA, Kmeans, Apriori Algorithm, Machine Learning, Data Mining, Association rule mining

I. Introduction

Market basket analysis is one of the data mining methods focusing on discovering purchasing patterns by extracting associations or co-occurrences from a store's transactional data. Market basket analysis determines the products which are bought together and to reorganize the supermarket layout, and also to design promotional campaigns such that products purchase can be improved. Hence, the Market consumer behaviors need to be analyzed, which can be done through different data mining techniques.

Informed decision can be made easily about product placement, pricing, promotion, profitability and also finds out, if there are any successful products that have no significant related elements. Similar products can be found so those can be placed near each other or it can be cross-sold.

A retailer must know the needs of customers and adapt to them. Market basket analysis is one possible way to find out which items can be put together. Market basket analyses gives retailer good information about related sales on group of goods basis Customers who buy s bread often also buy several products related to bread like milk, butter or jam. It makes sense that these groups are placed side by side in a retail center so that customers can access them quickly. Such related groups of goods also must be located side-by-side in order to remind customers of related items and to lead them through the center in a logical manner.

Principal Component analysis is a statistical technique to identify underling linear patterns in a data set so it can be expressed in terms of other data set of significantly lower dimension without much loss of information. The final data set should be able to explain most of the variance of the original dataset by making a variable reduction. The final variables are known as Principal Components.

K-means is a clustering/machine learning algorithm used to cluster observations into groups of related observations without any prior knowledge of those relationships. The k-means algorithm is one of the simplest clustering technique. The k-means algorithm clusters observations into k groups, where k is provided as an input parameter. It then assigns each observation to clusters based upon the observation's proximity to the mean of the cluster. It is used to find similar purchasing patterns and behaviors in market basket analysis

Association rules can be mined and this process of mining the association rules is one of the most important and powerful aspect of data mining. One of the main criteria of ARM is to find the relationship among various items in a database.

An association rule is of the form A→B where A is the antecedent and B is the consequent and here A and B are item sets and the underlying rule says us purchased by the customers who purchase A are likely to purchase B with a probability percentage factor as %C where C is known as confidence such a rule is as follows: "seventy percent of people who purchase beer will also like to purchase diapers" This helps the shop managers to study the behaviour or buying habits of the customers to increase the sales based on this study items that are regularly purchased by the customers are put under closed proximity. For example persons who purchase milk will also likely to purchase Bread.

The interestingness measures like support and confidence also plays a vital role in the association analysis. The support is defined as percentage of transactions that contained in the rule and is given by Support = (# of transactions involving A and B) / (total number of transactions).

The other factor is confidence it is the percentage of transactions that contain B if they contain A

Confidence = Probability(B if A) = P(B/A)

Confidence = (transactions involving A and B) / (total number of transactions that have A

In [ ]:
```
INSTACART MARKET BASKET ANALYSIS
```

In [1]:
```
#import the needed librairies

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
%matplotlib inline
import matplotlib.pyplot as plt  # Matlab-style plotting
import seaborn as sns
color = sns.color_palette()
import warnings
warnings.filterwarnings('ignore') #Supress unnecessary warnings for readabilit
y and cleaner presentation
pd.set_option('display.float_format', lambda x: '%.3f' % x) #Limiting floats o
utput to 3 decimal points
```

In [2]:
```
aisles = pd.read_csv('C:\\Users\\Surya\\instacart\\aisles.csv')
aisles.head(5)
```

Out[2]:

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

In [10]:
```
print(aisles.describe(), aisles.columns,
aisles.dtypes,  aisles.shape)
```

```
       aisle_id
count   134.000
mean     67.500
std      38.827
min       1.000
25%      34.250
50%      67.500
75%     100.750
max     134.000 Index(['aisle_id', 'aisle'], dtype='object') aisle_id     int
64
aisle       object
dtype: object (134, 2)
```

In [65]:
```
departments = pd.read_csv('C:\\Users\\Surya\\instacart\\departments.csv')
departments.head(5)
```

Out[65]:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

In [24]:
```
print(departments.describe(), departments.columns,
departments.dtypes,  departments.shape)
```

```
       department_id
count         21.000
mean          11.000
std            6.205
min            1.000
25%            6.000
50%           11.000
75%           16.000
max           21.000 Index(['department_id', 'department'], dtype='object') d
epartment_id     int64
department       object
dtype: object (21, 2)
```

In [10]:
```
orderproductsprior = pd.read_csv('C:\\Users\\Surya\\instacart\\orderproductspr
ior.csv')
orderproductsprior.head(5)
```

In [31]:
```
print(orderproductsprior.describe(), orderproductsprior.columns,
orderproductsprior.dtypes,  orderproductsprior.shape)
```

```
          order_id   product_id  add_to_cart_order    reordered
count 32434489.000 32434489.000       32434489.000 32434489.000
mean   1710748.519    25576.338              8.351        0.590
std     987300.696    14096.689              7.127        0.492
min          2.000        1.000              1.000        0.000
25%     855943.000    13530.000              3.000        0.000
50%    1711048.000    25256.000              6.000        1.000
75%    2565514.000    37935.000             11.000        1.000
max    3421083.000    49688.000            145.000        1.000 Index(['order
_id', 'product_id', 'add_to_cart_order', 'reordered'], dtype='object') order_
id             int64
product_id          int64
add_to_cart_order    int64
reordered           int64
dtype: object (32434489, 4)
```

In [ ]:
```
orderproductstrain = pd.read_csv('C:\\Users\\Surya\\instacart\\orderproductstr
ain.csv')
orderproductstrain
```

In [4]:
```
print(orders.describe(), orders.columns,
orders.dtypes,  orders.shape)
```

```
           order_id      user_id  order_number   order_dow  order_hour_of_day  \
count 3421083.000 3421083.000   3421083.000 3421083.000        3421083.000
mean  1710542.000  102978.208        17.155       2.776             13.452
std    987581.740   59533.718        17.733       2.047              4.226
min         1.000       1.000         1.000       0.000              0.000
25%    855271.500   51394.000         5.000       1.000             10.000
50%   1710542.000  102689.000        11.000       3.000             13.000
75%   2565812.500  154385.000        23.000       5.000             16.000
max   3421083.000  206209.000       100.000       6.000             23.000

       days_since_prior_order
count             3214874.000
mean                   11.115
std                     9.207
min                     0.000
25%                     4.000
50%                     7.000
75%                    15.000
max                    30.000   Index(['order_id', 'user_id', 'eval_set', 'or
der_number', 'order_dow',
       'order_hour_of_day', 'days_since_prior_order'],
      dtype='object') order_id                    int64
user_id                     int64
eval_set                   object
order_number                int64
order_dow                   int64
order_hour_of_day           int64
days_since_prior_order    float64
dtype: object (3421083, 7)
```

In [ ]:
```
orders = pd.read_csv('C:\\Users\\Surya\\instacart\\orders.csv')
orders
```

In [37]:
```
orders.isnull().sum()
```

Out[37]:
```
order_id                       0
user_id                        0
eval_set                       0
order_number                   0
order_dow                      0
order_hour_of_day              0
days_since_prior_order    206209
dtype: int64
```

In [38]:
```
#mean = sum of data / len of data
orders['days_since_prior_order'].sum()
```

Out[38]:
```
35732798.0
```

In [39]:
```
orders['days_since_prior_order'] = orders['days_since_prior_order'].fillna('1
0.444')
orders['days_since_prior_order']
```

In [26]:
```
print(orders.describe(), orders.columns,
orders.dtypes,  orders.shape)
```

```
          order_id     user_id  order_number   order_dow  order_hour_of_day
count 3421083.000 3421083.000   3421083.000 3421083.000        3421083.000
mean  1710542.000  102978.208        17.155       2.776             13.452
std    987581.740   59533.718        17.733       2.047              4.226
min         1.000       1.000         1.000       0.000              0.000
25%    855271.500   51394.000         5.000       1.000             10.000
50%   1710542.000  102689.000        11.000       3.000             13.000
75%   2565812.500  154385.000        23.000       5.000             16.000
max   3421083.000  206209.000       100.000       6.000             23.000 In
dex(['order_id', 'user_id', 'eval_set', 'order_number', 'order_dow',
       'order_hour_of_day', 'days_since_prior_order'],
      dtype='object') order_id                     int64
user_id                    int64
eval_set                  object
order_number               int64
order_dow                  int64
order_hour_of_day          int64
days_since_prior_order    object
dtype: object (3421083, 7)
```

In [96]:
```
# combine aisles, departments and products (left joined to products)
goods = pd.merge(left=pd.merge(left=products, right=departments, how='left'),
right=aisles, how='left')
# to retain '-' and make product names more "standard"
goods.product_name = goods.product_name.str.replace(' ', '_').str.lower()

goods.head()
```

Out[96]:
```
   product_id                                       product_name  aisle_id  \
0           1                          chocolate_sandwich_cookies        61
1           2                                    all-seasons_salt       104
2           3                robust_golden_unsweetened_oolong_tea        94
3           4  smart_ones_classic_favorites_mini_rigatoni_wit...        38
4           5                           green_chile_anytime_sauce         5

   department_id department                       aisle
0             19     snacks               cookies cakes
1             13     pantry          spices seasonings
2              7  beverages                         tea
3              1     frozen                frozen meals
4             13     pantry  marinades meat preparation
```

In [ ]:

In [20]:
```
order_products_all = pd.concat([orderproductstrain, orderproductsprior],
axis=0)

print("The order_products_all size is : ", order_products_all.shape)
```

```
The order_products_all size is :  (33819106, 4)
```

In [ ]:
```
te = all[all['eval_set']=='test']
te
```

In [63]:
```
order_products_all.head(5)
```

Out[63]:

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

In [22]:
```
all = pd.concat([alldata, orders], axis=0)

print("all size is : ", all.shape)
```

```
all size is :  (37289877, 13)
```

In [ ]:
```

```

In [41]:
```
departments.isnull().sum()
```

Out[41]:
```
department_id    0
department       0
dtype: int64
```

In [42]:
```
orders.isnull().sum()
```

Out[42]:
```
order_id                  0
user_id                   0
eval_set                  0
order_number              0
order_dow                 0
order_hour_of_day         0
days_since_prior_order    0
dtype: int64
```

In [40]:
```
aisles.isnull().sum()
```

Out[40]:
```
aisle_id    0
aisle       0
dtype: int64
```

In [44]:

In [38]:
```
products.isnull().sum()
```

Out[38]:
```
product_id       0
product_name     0
aisle_id         0
department_id    0
dtype: int64
```

In [57]:

In [46]:
```
total = order_products_all.isnull().sum().sort_values(ascending=False)
percent =
(order_products_all.isnull().sum()/order_products_all.isnull().count()).sort_v
alues(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total Missing', 'Per
cent'])
missing_data
```

Out[46]:

	Total Missing	Percent
reordered	0	0.000
add_to_cart_order	0	0.000
product_id	0	0.000
order_id	0	0.000

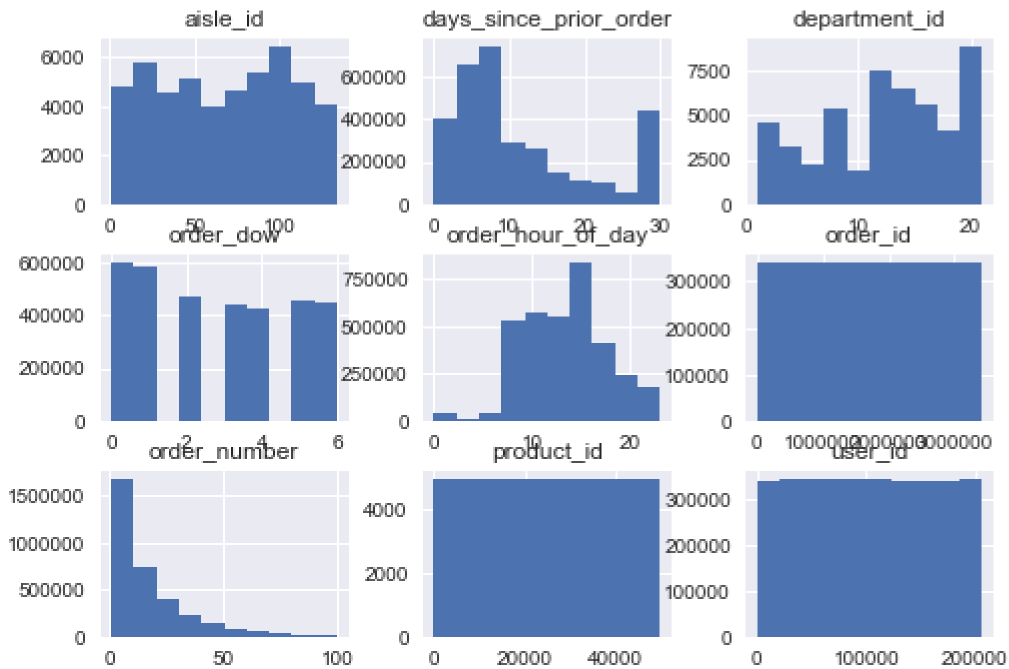
In [ ]:
```
There is no missing data in orderproductsprior and orderproductstrain
```

In [10]:
```
allpro = pd.concat([orders, products], axis=0)

print("all size is : ", allpro.shape)
allpro.hist()
```

```
all size is :  (3470771, 11)
```

Out[10]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001374D544CC0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000137003C1BE0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000013700438128
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000137004863C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000137004EAEF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000137004EAF28
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000137005AE400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000013700605710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000013700668F60
>]], dtype=object)
```

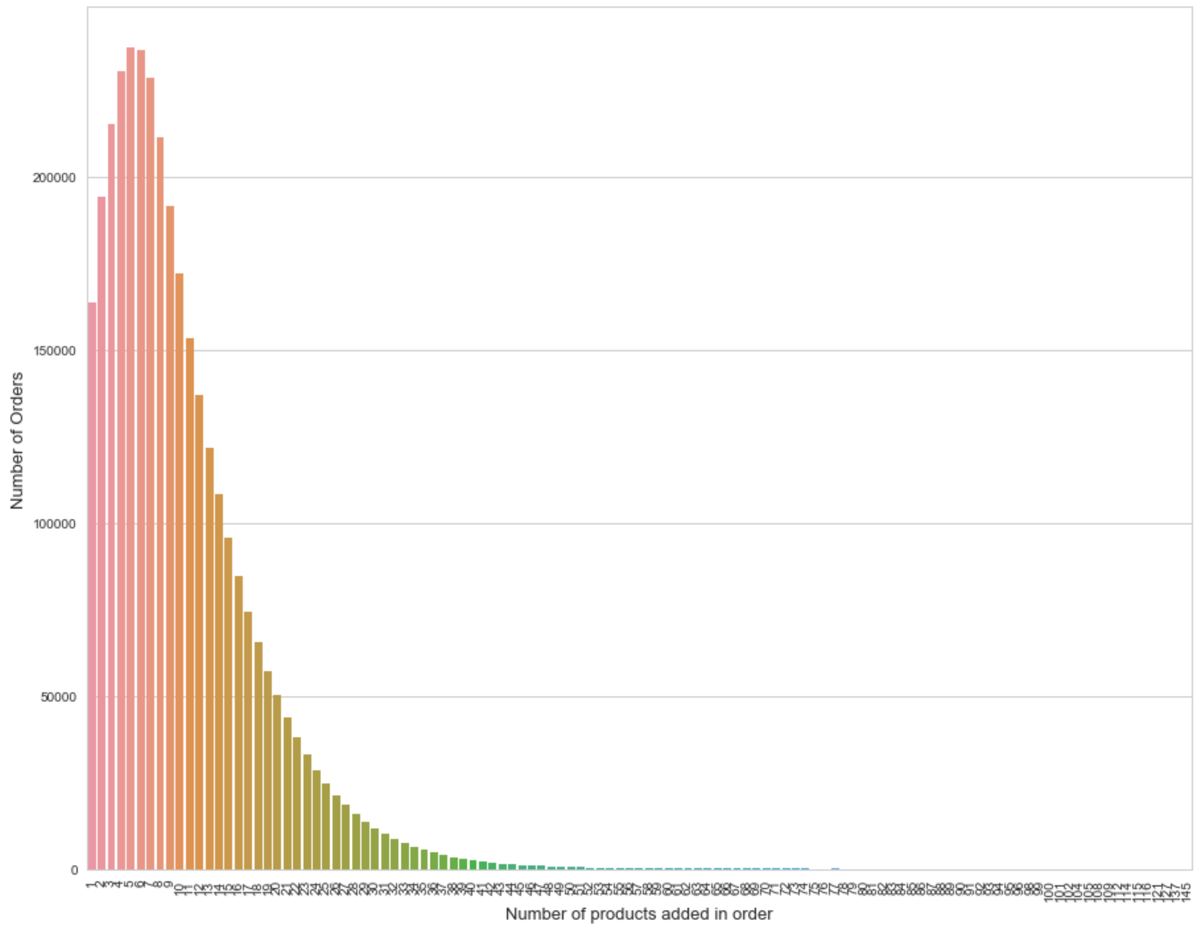


In [ ]:
```
Number of products that people usually order :
```

In [47]:
```
grouped = order_products_all.groupby("order_id")["add_to_cart_order"].aggregat
e("max").reset_index()
grouped = grouped.add_to_cart_order.value_counts()

sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)

plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('Number of products added in order', fontsize=13)
plt.show()
```



In [ ]:
```
We can observe that people usually order around 5 products.
```

In [ ]:
```
Most ordered Products
```

In [48]:
```
grouped = order_products_all.groupby("product_id")["reordered"].aggregate({'To
tal_reorders': 'count'}).reset_index()
grouped = pd.merge(grouped, products[['product_id', 'product_name']], how='lef
t', on=['product_id'])
grouped = grouped.sort_values(by='Total_reorders', ascending=False)[:10]
grouped
```

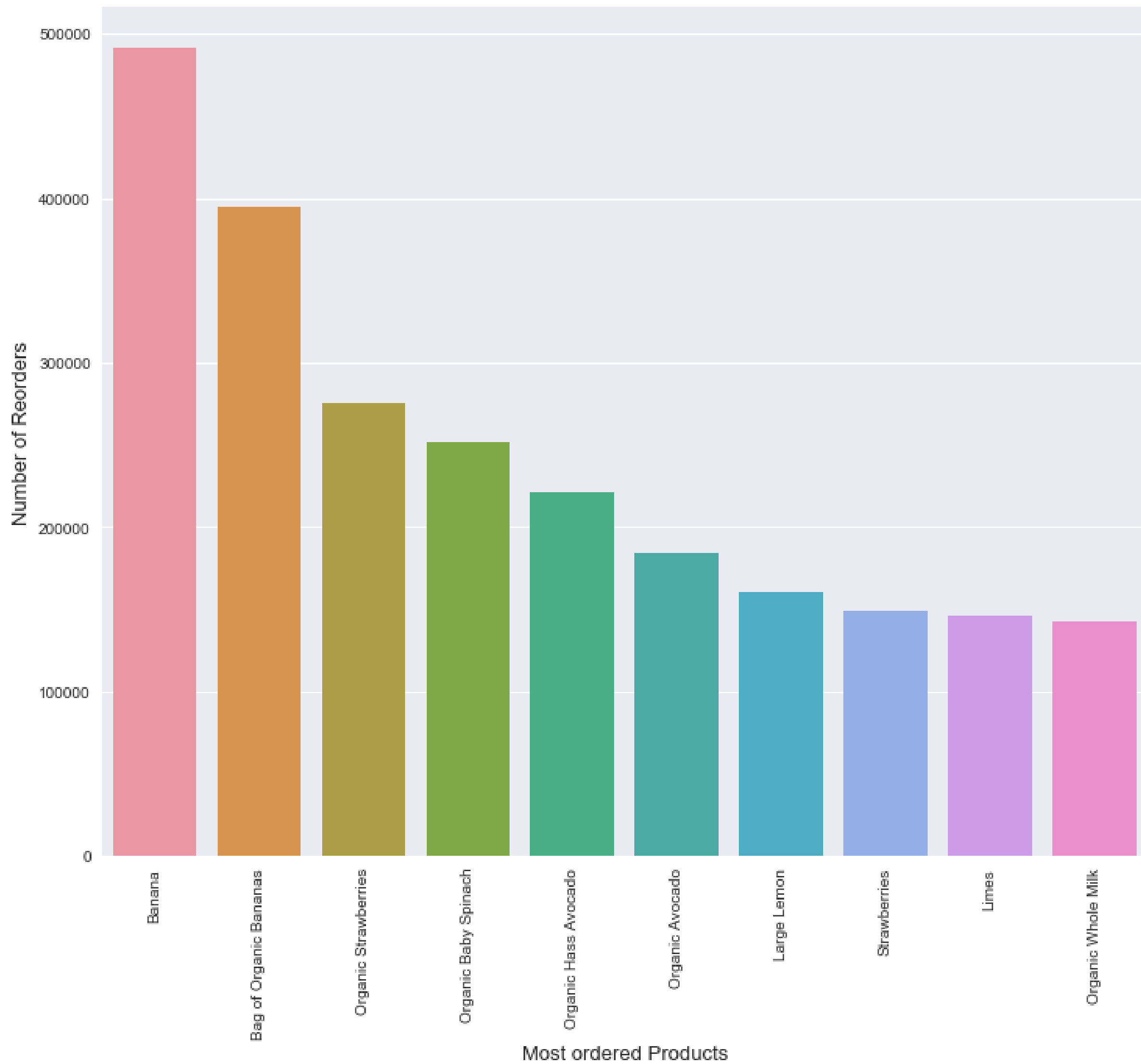
Out[48]:

	product_id	Total_reorders	product_name
24849	24852	491291	Banana
13173	13176	394930	Bag of Organic Bananas
21134	21137	275577	Organic Strawberries
21900	21903	251705	Organic Baby Spinach
47205	47209	220877	Organic Hass Avocado
47762	47766	184224	Organic Avocado
47622	47626	160792	Large Lemon
16794	16797	149445	Strawberries
26206	26209	146660	Limes
27842	27845	142813	Organic Whole Milk

In [ ]:
```
Fruits like banana , strawberries...are the most ordered products.
```

In [49]:
```
grouped  = grouped.groupby(['product_name']).sum()['Total_reorders'].sort_valu
es(ascending=False)

sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(12, 10))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of Reorders', fontsize=13)
plt.xlabel('Most ordered Products', fontsize=13)
plt.show()
```



In [ ]:
```
Reorder Frequency:
Do people usually reorder the same previous ordered products ?
```

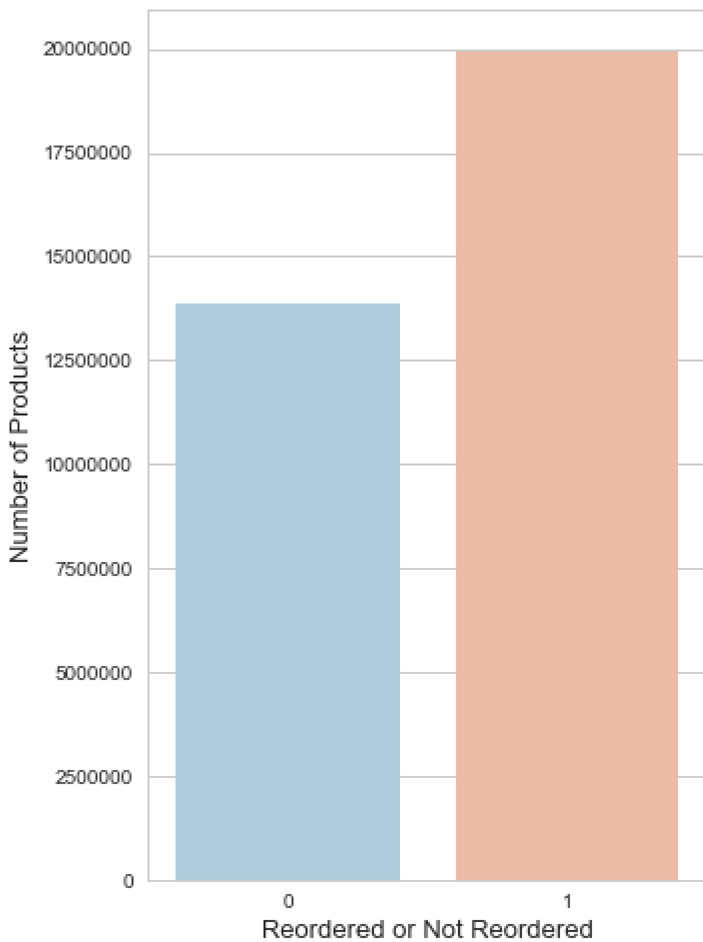
In [50]:
```
grouped = order_products_all.groupby("reordered")["product_id"].aggregate({'To
tal_products': 'count'}).reset_index()
grouped['Ratios'] = grouped["Total_products"].apply(lambda x: x /grouped['Tota
l_products'].sum())
grouped
```

Out[50]:

	reordered	Total_products	Ratios
0	0	13863746	0.410
1	1	19955360	0.590

In [51]:
```
grouped  = grouped.groupby(['reordered']).sum()
['Total_products'].sort_values(ascending=False)

sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(5, 8))
sns.barplot(grouped.index, grouped.values, palette='RdBu_r')
plt.ylabel('Number of Products', fontsize=13)
plt.xlabel('Reordered or Not Reordered', fontsize=13)
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```



In [ ]:
```
Most Reordered Products
Which products are usually reordered ?
```

In [52]:
```
grouped = order_products_all.groupby("product_id")["reordered"].aggregate({'re
order_sum': sum,'reorder_total': 'count'}).reset_index()
grouped['reorder_probability'] = grouped['reorder_sum'] / grouped['reorder_tot
al']
grouped = pd.merge(grouped, products[['product_id', 'product_name']], how='lef
t', on=['product_id'])
grouped = grouped[grouped.reorder_total > 75].sort_values(['reorder_probabilit
y'], ascending=False)[:10]
grouped
```

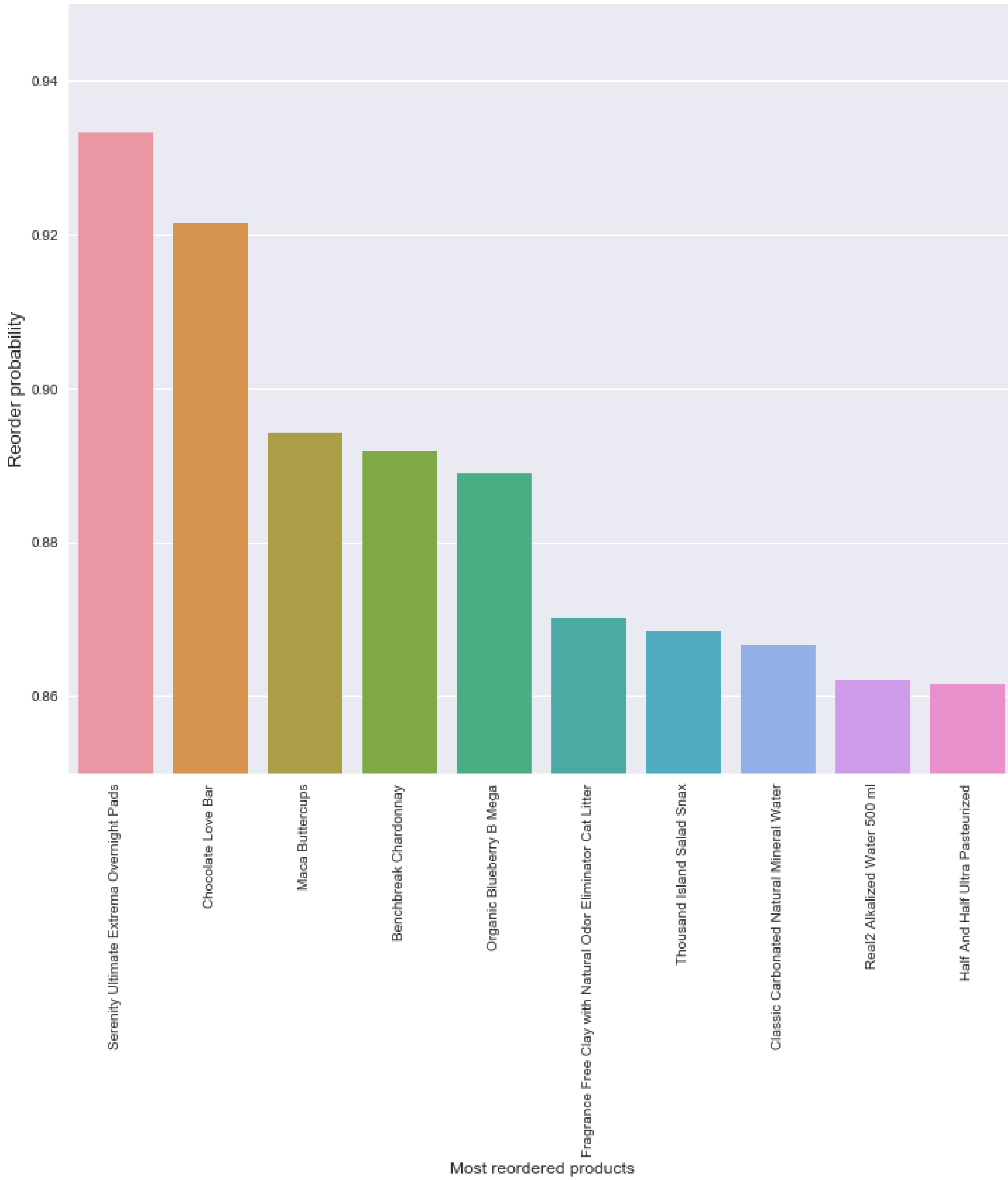
Out[52]:

	product_id	reorder_sum	reorder_total	reorder_probability	product_name
2074	2075	84	90	0.933	Serenity Ultimate Extrema Overnight Pads
27737	27740	94	102	0.922	Chocolate Love Bar
35601	35604	93	104	0.894	Maca Buttercups
38248	38251	99	111	0.892	Benchbreak Chardonnay
36798	36801	88	99	0.889	Organic Blueberry B Mega
10233	10236	114	131	0.870	Fragrance Free Clay with Natural Odor Eliminat...
20595	20598	99	114	0.868	Thousand Island Salad Snax
5455	5457	78	90	0.867	Classic Carbonated Natural Mineral Water
35493	35496	394	457	0.862	Real2 Alkalized Water 500 ml
9289	9292	2580	2995	0.861	Half And Half Ultra Pasteurized

In [ ]:
```
Serenity Ultimate Extrema Overnight Pads, Chocolate Love Bar, ....are the most
 reordered products
```

In [53]:
```
grouped  = grouped.groupby(['product_name']).sum()['reorder_probability'].sort
_values(ascending=False)

sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(12, 10))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylim([0.85,0.95])
plt.ylabel('Reorder probability', fontsize=13)
plt.xlabel('Most reordered products', fontsize=12)
plt.show()
```



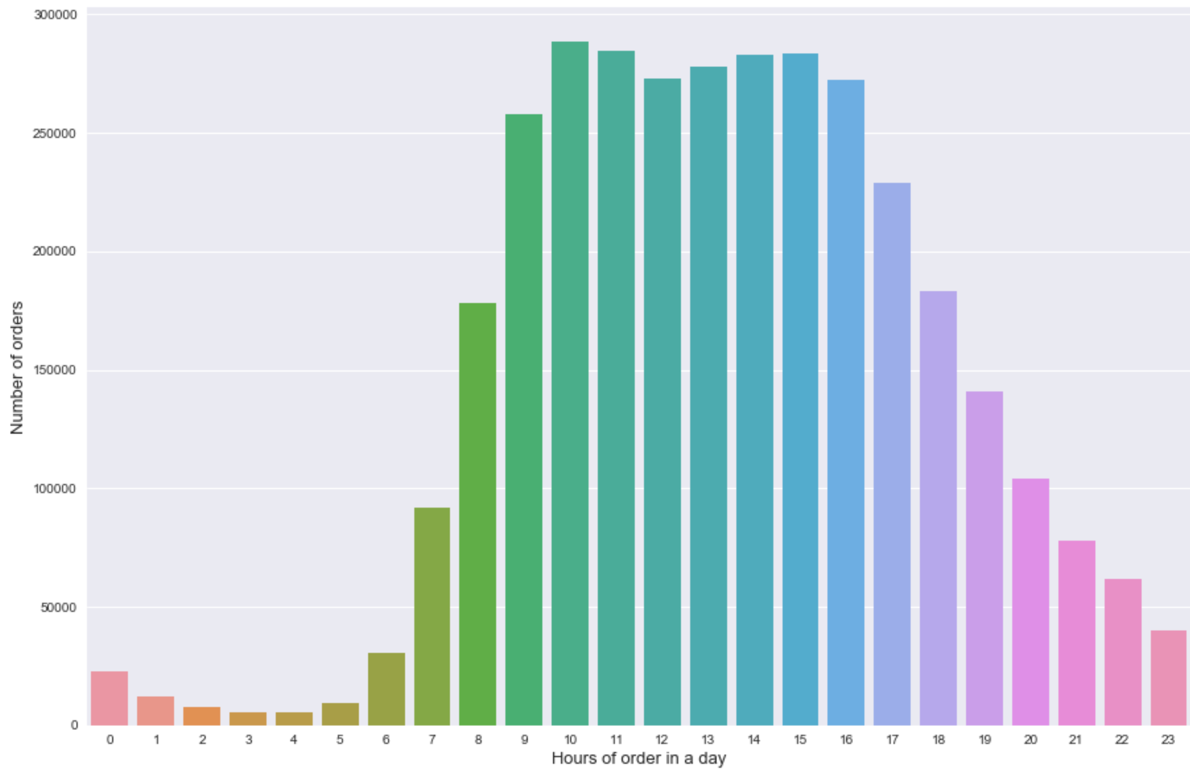
In [ ]:

```
Time of orders
Time at which people usually order products.
Hours of Order in a Day:
```

In [54]:

```
grouped = orders.groupby("order_id")["order_hour_of_day"].aggregate("sum").res
et_index()
grouped = grouped.order_hour_of_day.value_counts()

sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Hours of order in a day', fontsize=13)
plt.show()
```



In [ ]:

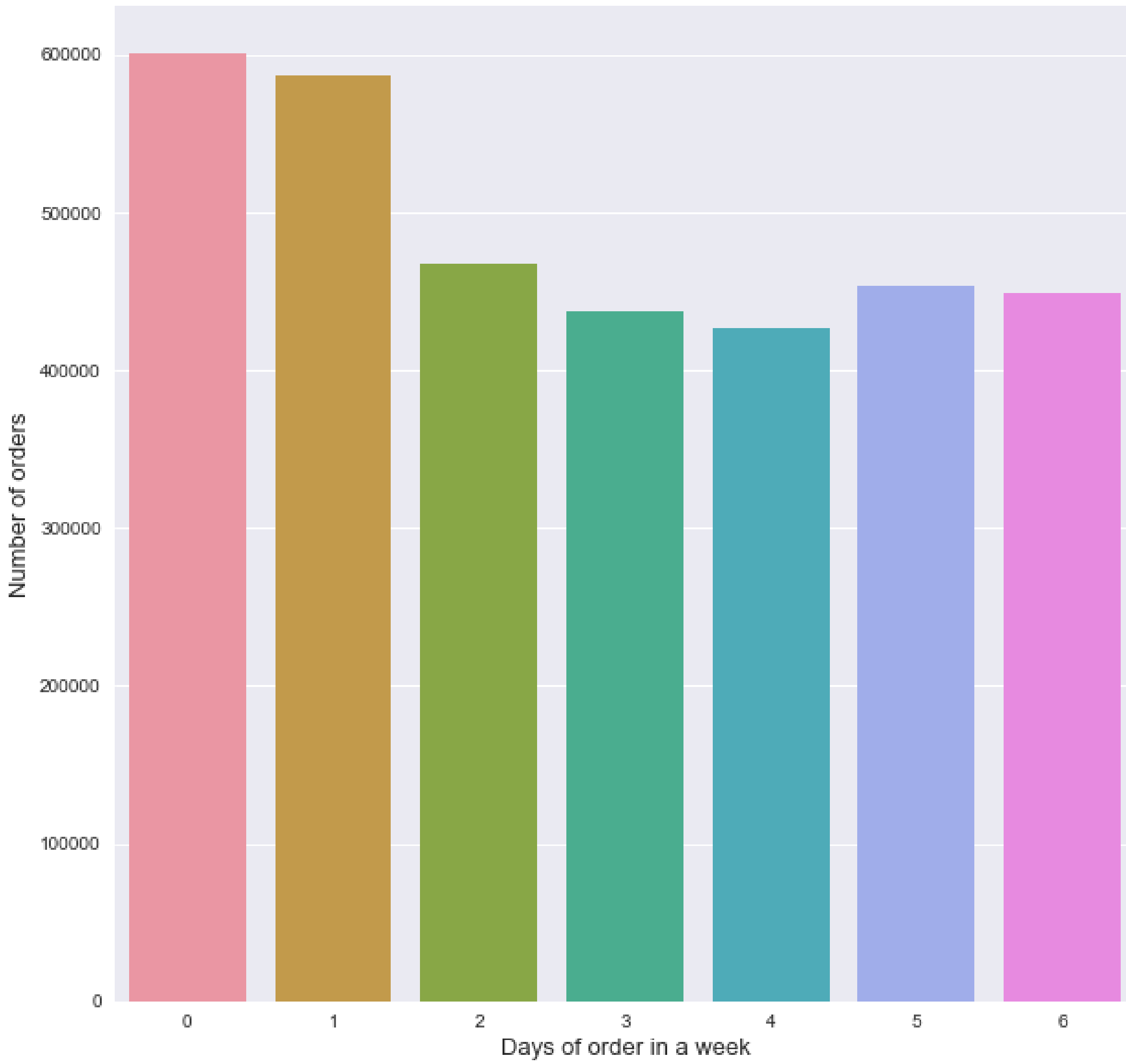
```
People mostly order from 7 a.m onwards
```

In [ ]:

```
Days of Orders in a week:
```

In [55]:
```
grouped = orders.groupby("order_id")
["order_dow"].aggregate("sum").reset_index()
grouped = grouped.order_dow.value_counts()

f, ax = plt.subplots(figsize=(10, 10))
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Days of order in a week', fontsize=13)
plt.show()
```



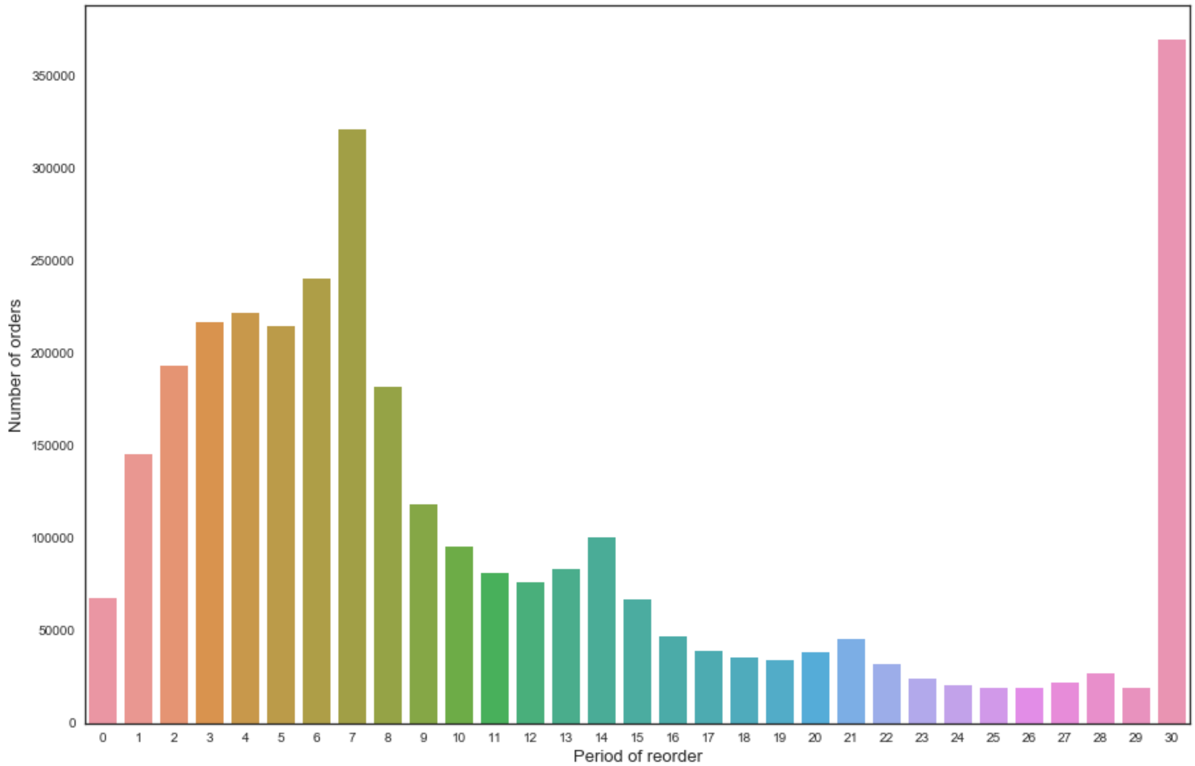
In [ ]:
```
People usually order at days 0 and 1 (anonymized days and probably the week en
d)
```

In [ ]:

In [ ]:
```
Period of Reorders:
```

In [91]:
```
grouped = orders.groupby("order_id")
["days_since_prior_order"].aggregate("sum").reset_index()
grouped = grouped.days_since_prior_order.value_counts()

from matplotlib.ticker import FormatStrFormatter
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(grouped.index, grouped.values)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.0f'))
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Period of reorder', fontsize=13)
plt.show()
```



In [ ]:
```
People usually reorder either from end of week or from end of the month.
```

In [ ]:
```
Orders in the whole dataset
Number and ratio of orders from the three datasets (prior, train, test).
```

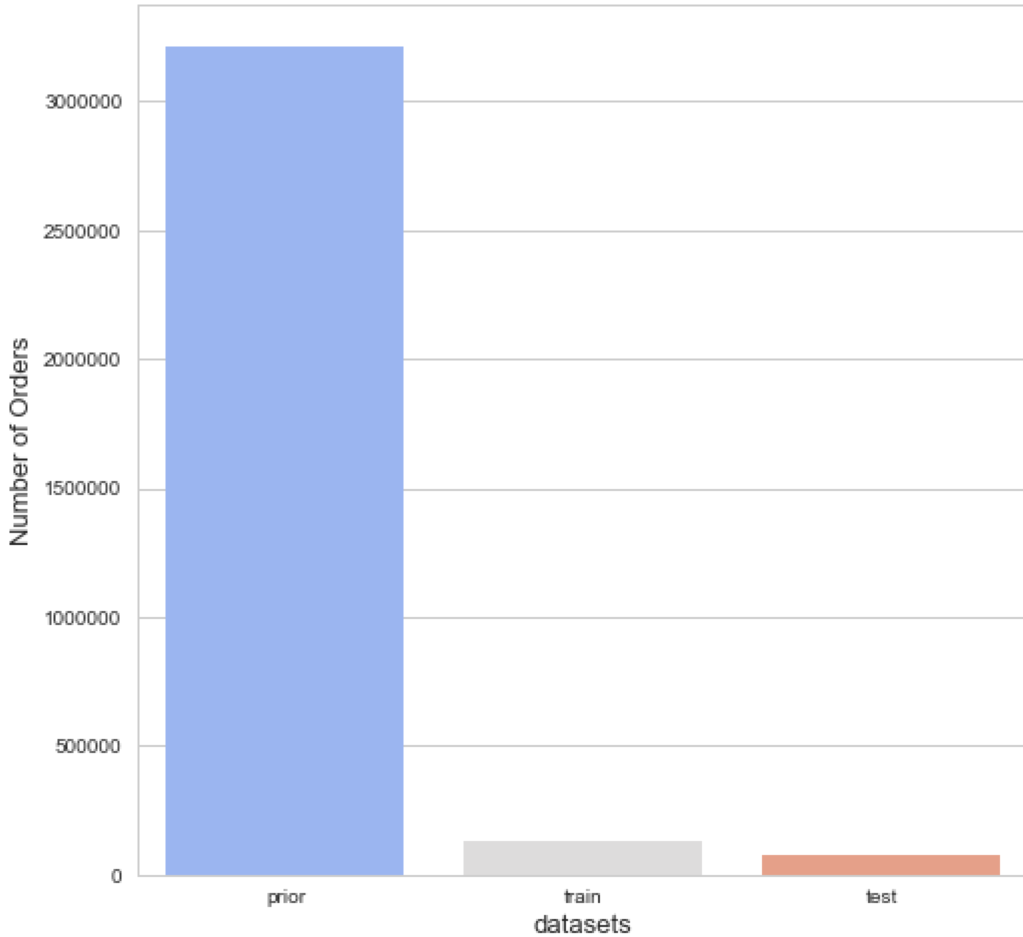
In [92]:
```
grouped = orders.groupby("eval_set")["order_id"].aggregate({'Total_orders': 'c
ount'}).reset_index()
grouped['Ratio'] = grouped["Total_orders"].apply(lambda x: x /grouped['Total_o
rders'].sum())
grouped
```

Out[92]:

	eval_set	Total_orders	Ratio
0	prior	3214874	0.940
1	test	75000	0.022
2	train	131209	0.038

In [49]:
```
grouped  = grouped.groupby(['eval_set']).sum()['Total_orders'].sort_values(asc
ending=False)

sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(8, 8))
sns.barplot(grouped.index, grouped.values, palette='coolwarm')
plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('datasets', fontsize=13)
plt.show()
```



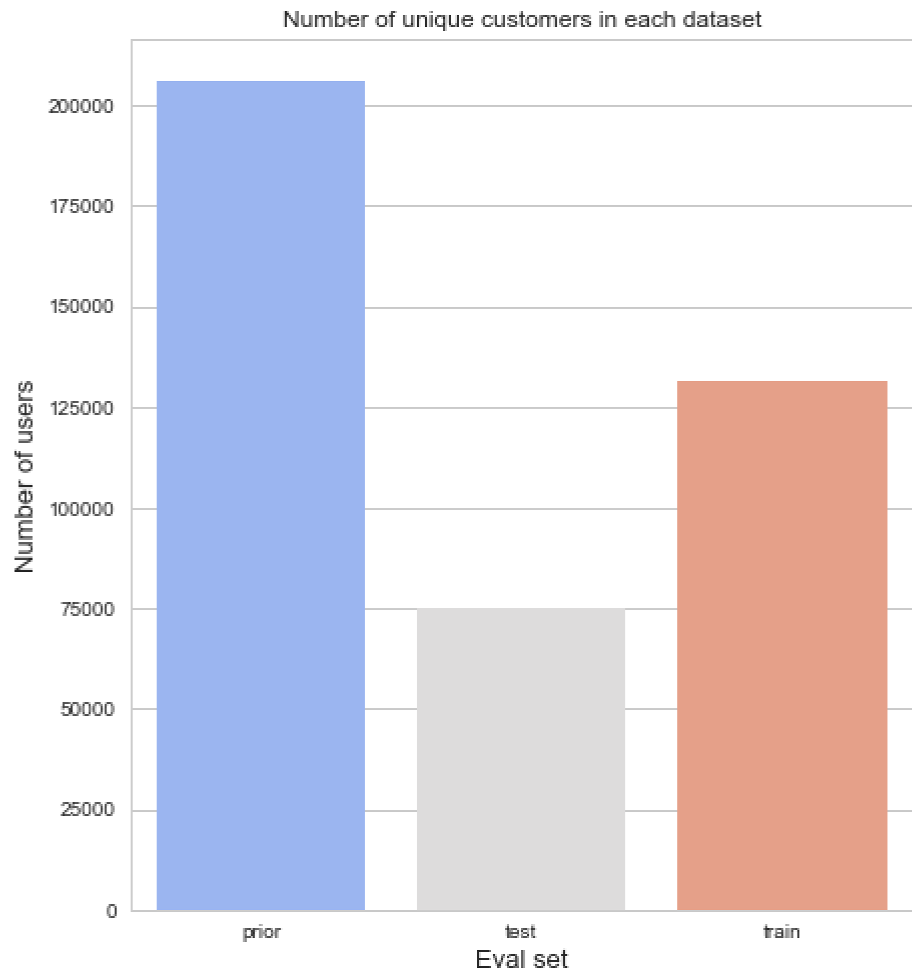
In [ ]:
```
Customers in the whole dataset
Let's check the total number of unique customers in the three datasets (prior,
 train, test).
```

In [50]:
```
print("Number of unique customers in the whole dataset : ",len(set(orders.user
_id)))
```

```
Number of unique customers in the whole dataset :  206209
```

In [51]:
```
grouped = orders.groupby("eval_set")["user_id"].apply(lambda x:
len(x.unique()))

plt.figure(figsize=(7,8))
sns.barplot(grouped.index, grouped.values, palette='coolwarm')
plt.ylabel('Number of users', fontsize=13)
plt.xlabel('Eval set', fontsize=13)
plt.title("Number of unique customers in each dataset")
plt.show()
```

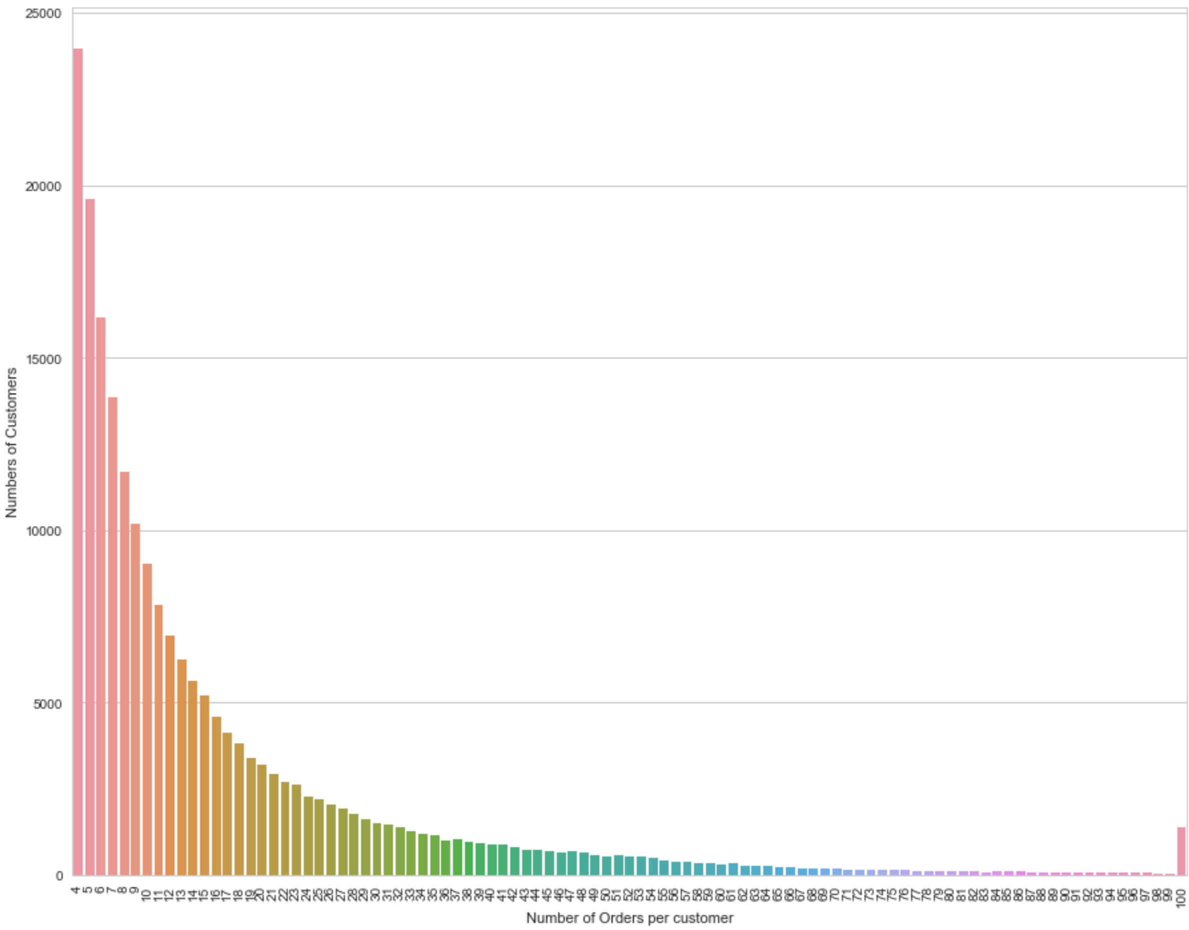


In [ ]:
```
Orders made by each customer
Let's check the number of orders made by each costumer in the whole dataset.
```

In [ ]:
```
```

In [5]:
```
grouped = orders.groupby('user_id')['order_id'].apply(lambda x:
len(x.unique())).reset_index()
grouped = grouped.groupby('order_id').aggregate("count")

sns.set_style("whitegrid")
f, ax = plt.subplots(figsize=(15, 12))
sns.barplot(grouped.index, grouped.user_id)
plt.ylabel('Numbers of Customers')
plt.xlabel('Number of Orders per customer')
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:
```
We can observe that most customers made 4 orders.
```

In [ ]:
```
Most important Departments (by number of products)
```

In [17]:

```
grouped = goods.groupby("department")["product_id"].aggregate({'Total_product
s': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_products"].apply(lambda x: x /grouped['Total
_products'].sum())
grouped.sort_values(by='Total_products', ascending=False, inplace=True)
grouped
```

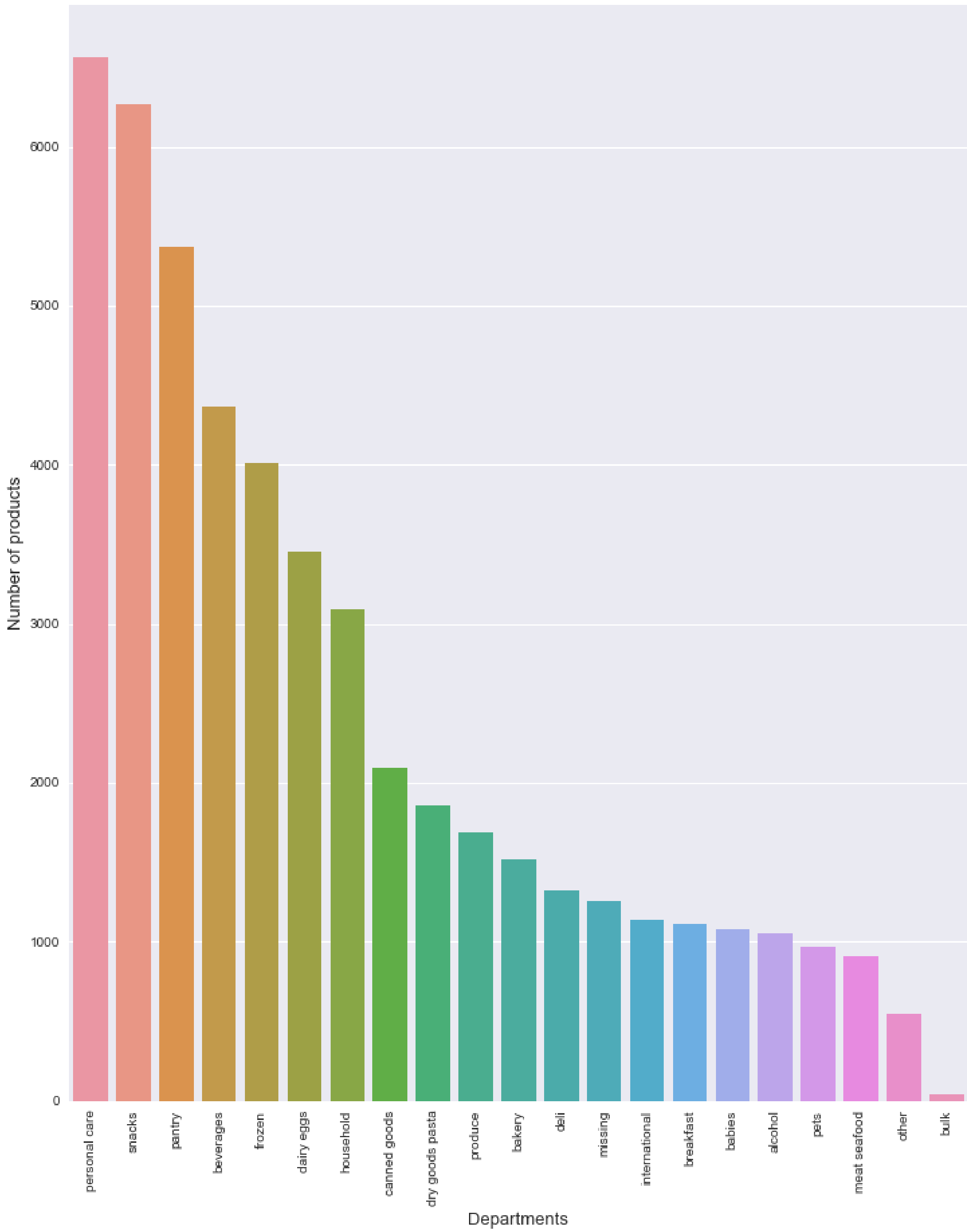
Out[17]:

	department	Total_products	Ratio
17	personal care	6563	0.132
20	snacks	6264	0.126
16	pantry	5371	0.108
3	beverages	4365	0.088
10	frozen	4007	0.081
7	dairy eggs	3449	0.069
11	household	3085	0.062
6	canned goods	2092	0.042
9	dry goods pasta	1858	0.037
19	produce	1684	0.034
2	bakery	1516	0.031
8	deli	1322	0.027
14	missing	1258	0.025
12	international	1139	0.023
4	breakfast	1115	0.022
1	babies	1081	0.022
0	alcohol	1054	0.021
18	pets	972	0.020
13	meat seafood	907	0.018
15	other	548	0.011
5	bulk	38	0.001

In [ ]:

In [18]:
```
grouped  = grouped.groupby(['department']).sum()
['Total_products'].sort_values(ascending=False)

sns.set_style("darkgrid")
f, ax = plt.subplots(figsize=(12, 15))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of products', fontsize=13)
plt.xlabel('Departments', fontsize=13)
plt.show()
```



In [ ]:

In [159]:
```
order_prior = pd.merge(orderproductsprior,orders,on=['order_id','order_id'])
order_prior = order_prior.sort_values(by=['user_id','order_id'])
order_prior.head()
```

Out[159]:
```
         order_id  product_id  add_to_cart_order  reordered  user_id eval_set  \
4089398    431534         196                  1          1        1    prior
4089399    431534       12427                  2          1        1    prior
4089400    431534       10258                  3          1        1    prior
4089401    431534       25133                  4          1        1    prior
4089402    431534       10326                  5          0        1    prior

         order_number  order_dow  order_hour_of_day  days_since_prior_order
4089398             5          4                 15                  28.000
4089399             5          4                 15                  28.000
4089400             5          4                 15                  28.000
4089401             5          4                 15                  28.000
4089402             5          4                 15                  28.000
```

In [ ]:

In [14]:
```
_mt = pd.merge(order_products_all,orders,on=['order_id','order_id'])
_mt = pd.merge(_mt,products,on=['product_id','product_id'])
mt = pd.merge(_mt,aisles,on=['aisle_id','aisle_id'])
mt.head(10)
```

Out[14]:

	order_id	product_id	add_to_cart_order	reordered	user_id	eval_set	order_number
0	1	49302	1	1	112108	train	4
1	816049	49302	7	1	47901	train	14
2	1242203	49302	1	1	2993	train	15
3	1383349	49302	11	1	41425	train	4
4	1787378	49302	8	0	187205	train	5
5	2445303	49302	2	1	199120	train	49
6	2853065	49302	12	1	145852	train	7
7	3231517	49302	6	1	63189	train	42
8	38841	49302	5	1	139875	prior	3
9	45900	49302	19	0	16919	prior	8

In [26]:
```
mt['eval_set'].value_counts()
```

Out[26]:
```
prior    32434489
train     1384617
Name: eval_set, dtype: int64
```

In [ ]:

In [31]:
```
te['eval_set'].value_counts()
```

Out[31]:
```
test    75000
Name: eval_set, dtype: int64
```

In [29]:
```
mt['product_name'].value_counts()[0:10]
```

Out[29]:
```
Banana                      491291
Bag of Organic Bananas      394930
Organic Strawberries        275577
Organic Baby Spinach        251705
Organic Hass Avocado        220877
Organic Avocado             184224
Large Lemon                 160792
Strawberries                149445
Limes                       146660
Organic Whole Milk          142813
Name: product_name, dtype: int64
```

In [31]:
```
mt['aisle'].value_counts()[0:10]
```

Out[31]:
```
fresh fruits                     3792661
fresh vegetables                 3568630
packaged vegetables fruits       1843806
yogurt                           1507583
packaged cheese                  1021462
milk                              923659
water seltzer sparkling water     878150
chips pretzels                    753739
soy lactosefree                   664493
bread                             608469
Name: aisle, dtype: int64
```

In [ ]:
```
Fresh fruits and fresh vegetables are the best selling goods.
```

In [33]:
```
#Clustering Customers prior and train
cust_prod = pd.crosstab(mt['user_id'], mt['aisle'])
cust_prod.head(10)
```

Out[33]:

aisle	air fresheners candles	asian foods	baby accessories	baby bath body care	baby food formula	bakery desserts	baking ingredients	baking supplies decor
user_id								
1	0	0	0	0	0	0	0	0
2	0	3	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	1	4	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	3	0
8	0	1	0	0	0	0	1	0
9	0	0	0	0	6	0	2	0
10	0	1	0	0	0	0	0	0

10 rows × 134 columns

In [43]:
```
cust_prod.shape
```

Out[43]: (206209, 134)

In [35]:
```
te.shape
```

Out[35]: (75000, 13)

In [ ]:

In [49]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=6)
pca.fit(cust_prod)
pca_samples = pca.transform(cust_prod)
```

In [50]:
```
ps = pd.DataFrame(pca_samples)
ps.head()
```

Out[50]:

	0	1	2	3	4	5
0	-24.216	2.429	-2.466	-0.146	0.269	-1.433
1	6.463	36.751	8.383	15.098	-6.921	-0.979
2	-7.990	2.404	-11.030	0.672	-0.442	-2.823
3	-27.991	-0.756	-1.922	2.092	-0.288	0.926
4	-19.896	-2.637	0.533	3.679	0.613	-1.624

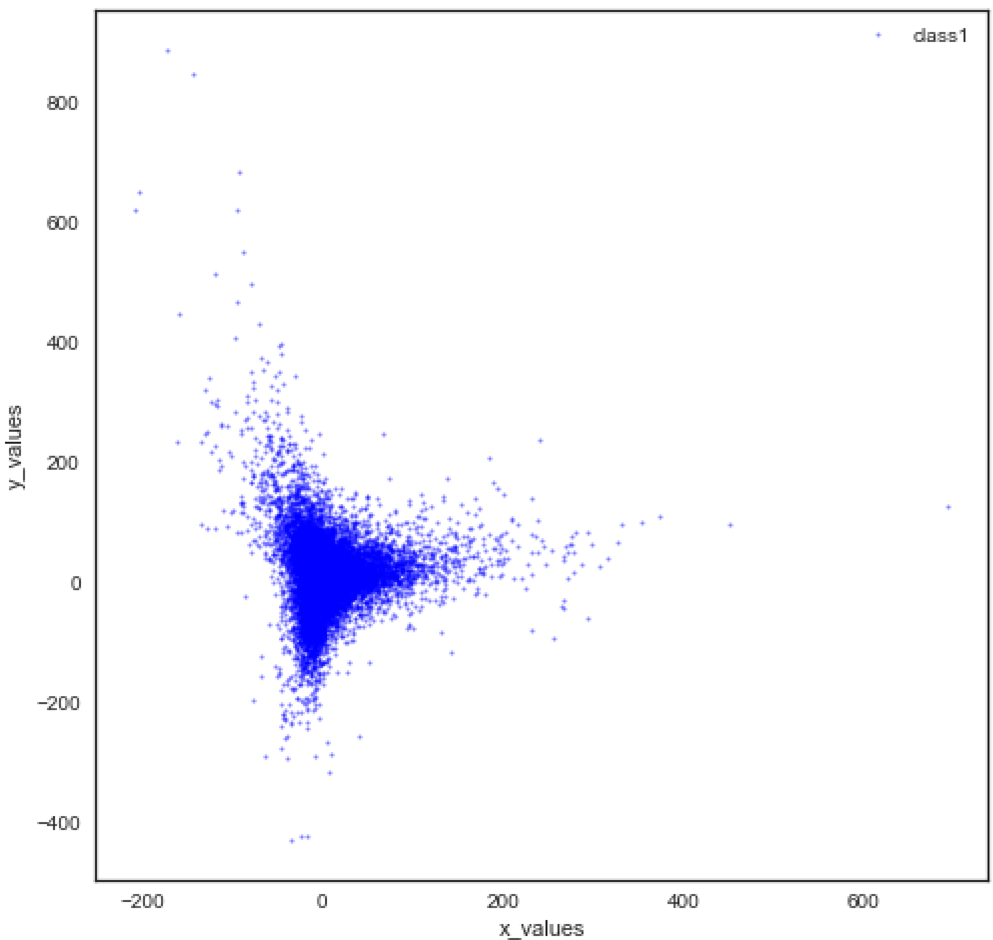
In [ ]:

In [51]:
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
tocluster = pd.DataFrame(ps[[4,1]])
print (tocluster.shape)
print (tocluster.head())

fig = plt.figure(figsize=(8,8))
plt.plot(tocluster[4], tocluster[1], 'o', markersize=2, color='blue', alpha=0.
5, label='class1')

plt.xlabel('x_values')
plt.ylabel('y_values')
plt.legend()
plt.show()
```

```
(206209, 2)
       4      1
0  0.269  2.429
1 -6.921 36.751
2 -0.442  2.404
3 -0.288 -0.756
4  0.613 -2.637
```



In [52]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

clusterer = KMeans(n_clusters=4,random_state=42).fit(tocluster)
centers = clusterer.cluster_centers_
c_preds = clusterer.predict(tocluster)
print(centers)
```

```
[[ -0.11868823   0.09644088]
 [-11.26759816  65.248165  ]
 [ -4.71388765 -40.63421033]
 [ 76.82338978  26.26358548]]
```

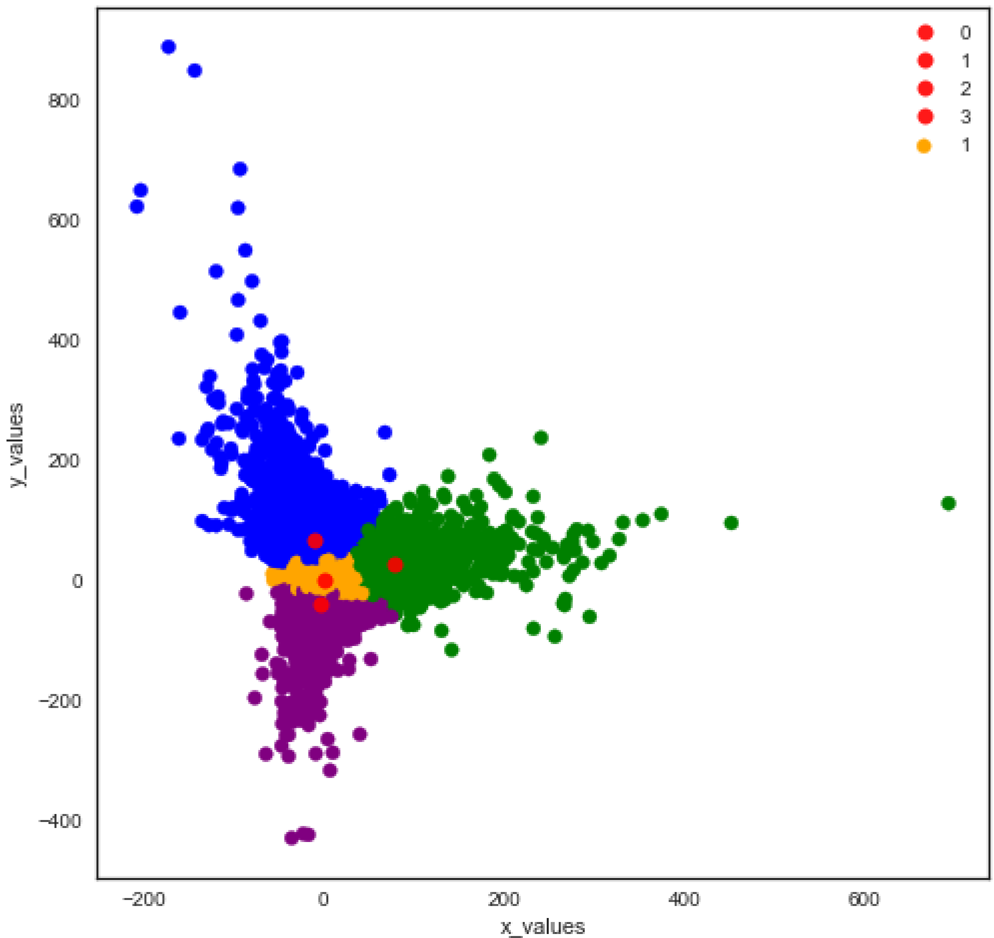
In [53]:
```
print (c_preds[0:100])
```

```
[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 1 0 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2 0 0 0 0 0 0 0 0 0 0 2 0 0 2 1 0 0 0 0 0 0 0 0 0 0]
```

In [55]:
```
import matplotlib
fig = plt.figure(figsize=(8,8))
colors = ['orange','blue','purple','green']
colored = [colors[k] for k in c_preds]
print (colored[0:10])
plt.scatter(tocluster[4],tocluster[1],  color = colored)
for ci,c in enumerate(centers):
    plt.plot(c[0], c[1], 'o', markersize=8, color='red', alpha=0.9, label=''+s
tr(ci))

plt.xlabel('x_values')
plt.ylabel('y_values')
plt.legend()
plt.show()
```

```
['orange', 'blue', 'orange', 'orange', 'orange', 'orange', 'orange', 'orang
e', 'orange', 'orange']
```

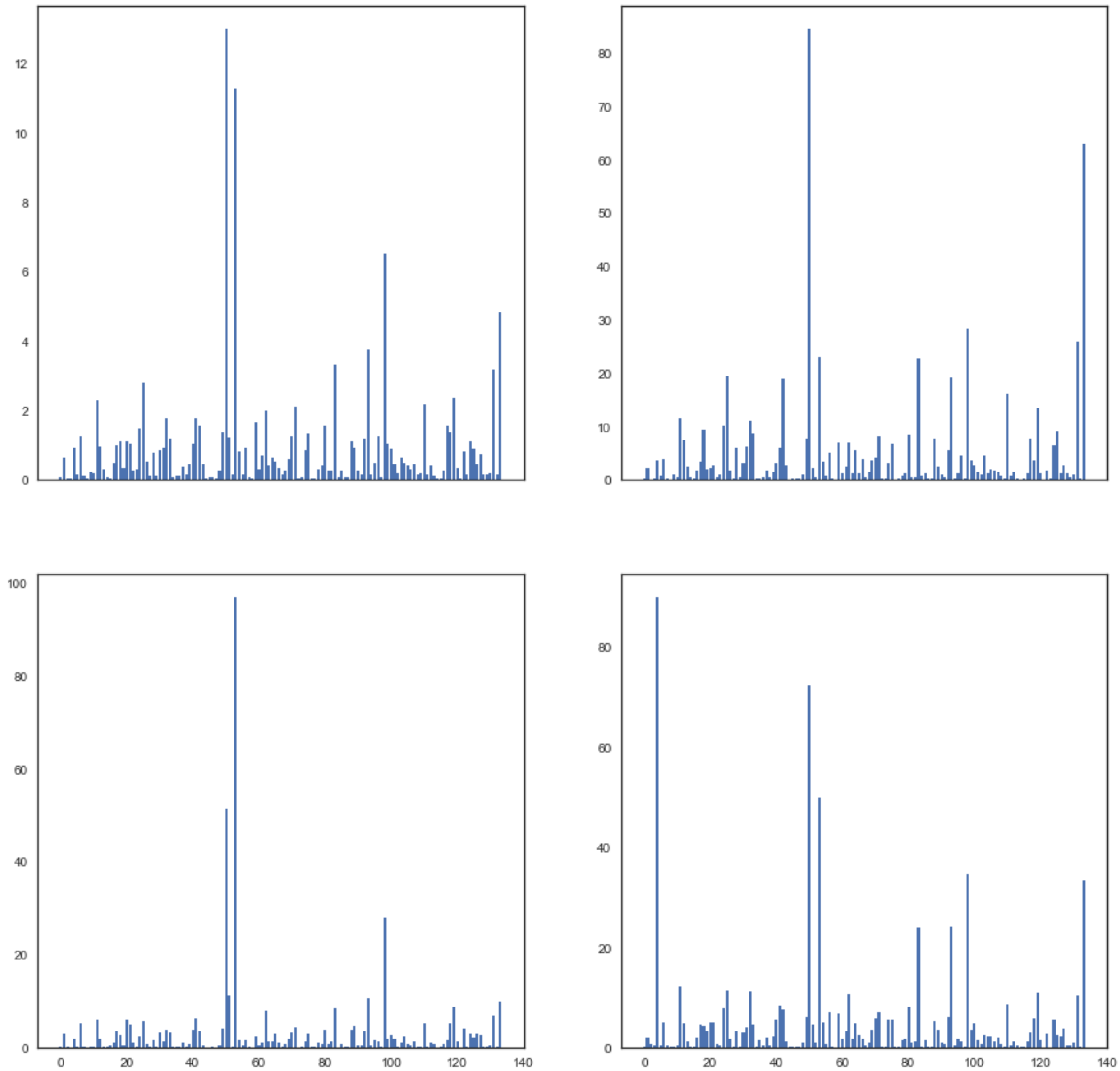


In [57]:
```
print (clust_prod.shape)
f,arr = plt.subplots(2,2,sharex=True,figsize=(15,15))

c1_count = len(clust_prod[clust_prod['cluster']==0])

c0 = clust_prod[clust_prod['cluster']==0].drop('cluster',axis=1).mean()
arr[0,0].bar(range(len(clust_prod.drop('cluster',axis=1).columns)),c0)
c1 = clust_prod[clust_prod['cluster']==1].drop('cluster',axis=1).mean()
arr[0,1].bar(range(len(clust_prod.drop('cluster',axis=1).columns)),c1)
c2 = clust_prod[clust_prod['cluster']==2].drop('cluster',axis=1).mean()
arr[1,0].bar(range(len(clust_prod.drop('cluster',axis=1).columns)),c2)
c3 = clust_prod[clust_prod['cluster']==3].drop('cluster',axis=1).mean()
arr[1,1].bar(range(len(clust_prod.drop('cluster',axis=1).columns)),c3)
plt.show()
```

```
(206209, 135)
```



In [58]:
```
c0.sort_values(ascending=False)[0:10]
```

Out[58]:
```
aisle
fresh fruits                     12.997
fresh vegetables                 11.265
packaged vegetables fruits        6.532
yogurt                            4.839
packaged cheese                   3.755
milk                              3.303
water seltzer sparkling water     3.169
chips pretzels                    2.783
soy lactosefree                   2.350
bread                             2.279
dtype: float64
```

In [59]:
```
c1.sort_values(ascending=False)[0:10]
```

Out[59]:
```
aisle
fresh fruits                     84.445
yogurt                           62.985
packaged vegetables fruits       28.129
water seltzer sparkling water    25.796
fresh vegetables                 22.892
milk                             22.727
chips pretzels                   19.450
packaged cheese                  19.043
energy granola bars              19.022
refrigerated                     16.013
dtype: float64
```

In [60]:
```
c2.sort_values(ascending=False)[0:10]
```

Out[60]:
```
aisle
fresh vegetables                 96.942
fresh fruits                     51.420
packaged vegetables fruits       27.925
fresh herbs                      11.318
packaged cheese                  10.646
yogurt                            9.926
soy lactosefree                   8.805
milk                              8.353
frozen produce                    7.815
water seltzer sparkling water     6.770
dtype: float64
```

In [61]:
```
c3.sort_values(ascending=False)[0:10]
```

Out[61]:
```
aisle
baby food formula           90.031
fresh fruits                72.334
fresh vegetables            50.059
packaged vegetables fruits  34.557
yogurt                      33.243
packaged cheese             24.305
milk                        23.997
bread                       12.201
chips pretzels              11.458
crackers                    11.248
dtype: float64
```

In [62]:
```
from IPython.display import display, HTML
cluster_means = [[c0['fresh fruits'],c0['fresh vegetables'],c0['packaged veget
ables fruits'], c0['yogurt'], c0['packaged cheese'], c0['milk'],c0['water selt
zer sparkling water'],c0['chips pretzels']],
                 [c1['fresh fruits'],c1['fresh vegetables'],c1['packaged veget
ables fruits'], c1['yogurt'], c1['packaged cheese'], c1['milk'],c1['water selt
zer sparkling water'],c1['chips pretzels']],
                 [c2['fresh fruits'],c2['fresh vegetables'],c2['packaged veget
ables fruits'], c2['yogurt'], c2['packaged cheese'], c2['milk'],c2['water selt
zer sparkling water'],c2['chips pretzels']],
                 [c3['fresh fruits'],c3['fresh vegetables'],c3['packaged veget
ables fruits'], c3['yogurt'], c3['packaged cheese'], c3['milk'],c3['water selt
zer sparkling water'],c3['chips pretzels']]]
cluster_means = pd.DataFrame(cluster_means, columns = ['fresh fruits','fresh v
egetables','packaged vegetables fruits','yogurt','packaged cheese','milk','wat
er seltzer sparkling water','chips pretzels'])
HTML(cluster_means.to_html())
```

Out[62]:

	fresh fruits	fresh vegetables	packaged vegetables fruits	yogurt	packaged cheese	milk	water seltzer sparkling water	chips pretzels
0	12.997	11.265	6.532	4.839	3.755	3.303	3.169	2.783
1	84.445	22.892	28.129	62.985	19.043	22.727	25.796	19.450
2	51.420	96.942	27.925	9.926	10.646	8.353	6.770	5.796
3	72.334	50.059	34.557	33.243	24.305	23.997	10.528	11.458

In [63]:
```
cluster_perc = cluster_means.iloc[:, :].apply(lambda x: (x / x.sum())*100,axis=1)
HTML(cluster_perc.to_html())
```

Out[63]:

	fresh fruits	fresh vegetables	packaged vegetables fruits	yogurt	packaged cheese	milk	water seltzer sparkling water	chips pretzels
0	26.720	23.158	13.429	9.948	7.719	6.791	6.514	5.721
1	29.582	8.019	9.854	22.064	6.671	7.961	9.036	6.813
2	23.611	44.514	12.823	4.558	4.888	3.836	3.109	2.661
3	27.769	19.218	13.267	12.762	9.331	9.212	4.042	4.399

In [64]:
```
c0.sort_values(ascending=False)[10:15]
```

Out[64]:
```
aisle
refrigerated     2.169
ice cream ice    2.083
frozen produce   2.001
eggs             1.778
crackers         1.766
dtype: float64
```

In [65]:
```
c1.sort_values(ascending=False)[10:15]
```

Out[65]:
```
aisle
soy lactosefree   13.437
bread             11.515
crackers          10.998
cereal             9.971
candy chocolate    9.348
dtype: float64
```

In [66]:
```
c2.sort_values(ascending=False)[10:15]
```

Out[66]:
```
aisle
eggs                          6.177
canned jarred vegetables   6.100
bread                         6.015
chips pretzels                5.796
refrigerated                  5.281
dtype: float64
```

In [67]:
```
c3.sort_values(ascending=False)[10:15]
```

Out[67]:
```
aisle
soy lactosefree                 11.003
frozen produce                  10.577
water seltzer sparkling water   10.528
refrigerated                     8.530
eggs                             8.318
dtype: float64
```

In [58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.sparse import csr_matrix
from collections import Counter
```

In [11]:
```
test = orders[orders['eval_set']=='test']
prior = orders[orders['eval_set']=='prior']
train= orders[orders['eval_set']=='train']
test.tail()
```

Out[11]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	day
3420918	2728930	206202	test	23	2	17	6.0C
3420929	350108	206204	test	5	4	14	14.C
3421001	1043943	206206	test	68	0	20	0.0C
3421018	2821651	206207	test	17	2	13	14.C
3421068	803273	206208	test	50	5	11	4.0C

In [12]:
```
len(test.index)
```

Out[12]: 75000

In [ ]:
```
Best Selling Products
```

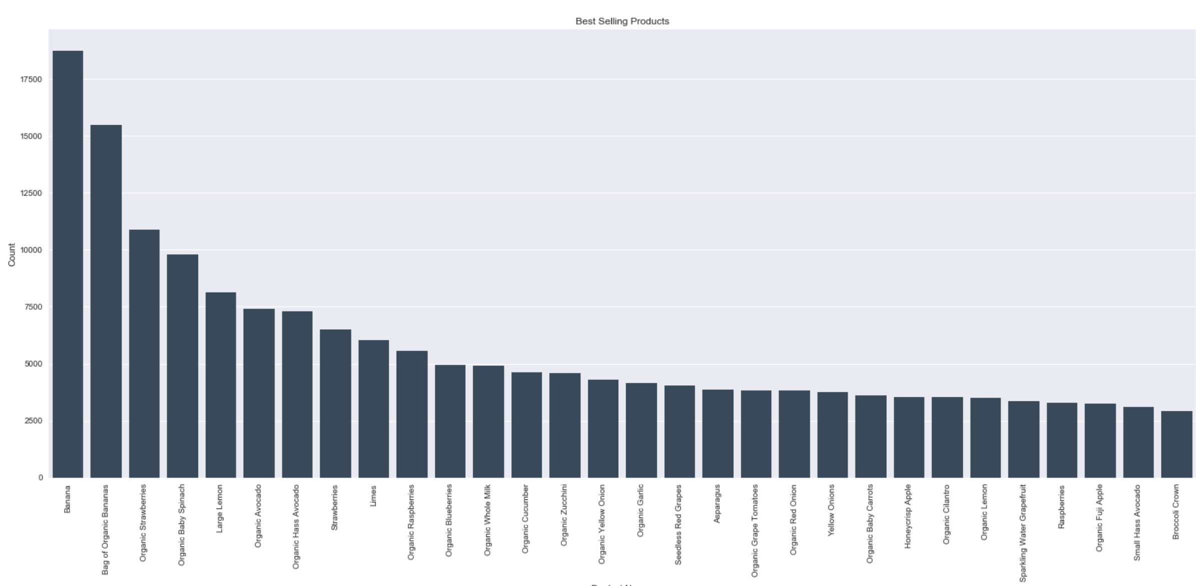
In [63]:
```
import seaborn as sn
productsCount = orderproductstrain["product_id"].value_counts().to_frame()
productsCount["count"] = productsCount.product_id
productsCount["product_id"] = productsCount.index
mergedData = pd.merge(productsCount,products,how="left",on="product_id").sort_
values(by="count",ascending=False)

fig,ax = plt.subplots()
fig.set_size_inches(25,10)
sn.barplot(data=mergedData.head(30),x="product_name",y="count",ax=ax,orient="v'
olor="#34495e")
ax.set(xlabel='Product Names',ylabel="Count",title="Best Selling Products")
plt.xticks(rotation=90)

mergedData.head(10)
```

Out[63]:

	product_id	count	product_name	aisle_id	department_id
0	24852	18726	Banana	24	4
1	13176	15480	Bag of Organic Bananas	24	4
2	21137	10894	Organic Strawberries	24	4
3	21903	9784	Organic Baby Spinach	123	4
4	47626	8135	Large Lemon	24	4
5	47766	7409	Organic Avocado	24	4
6	47209	7293	Organic Hass Avocado	24	4
7	16797	6494	Strawberries	24	4
8	26209	6033	Limes	24	4
9	27966	5546	Organic Raspberries	123	4



In [ ]:

In [23]:
```
from numpy import *
```

In [38]:
```
def createC1(te):
    C1 = []
    for transaction in te:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])

    C1.sort()
    return list(map(frozenset, C1))#use frozen set so we
                                #can use it as a key in a dict
```

In [39]:
```
C1 = createC1(te)
C1
```

Out[39]:
```
[frozenset({'_'}),
 frozenset({'a'}),
 frozenset({'b'}),
 frozenset({'c'}),
 frozenset({'d'}),
 frozenset({'e'}),
 frozenset({'f'}),
 frozenset({'h'}),
 frozenset({'i'}),
 frozenset({'l'}),
 frozenset({'m'}),
 frozenset({'n'}),
 frozenset({'o'}),
 frozenset({'p'}),
 frozenset({'r'}),
 frozenset({'s'}),
 frozenset({'t'}),
 frozenset({'u'}),
 frozenset({'v'}),
 frozenset({'w'}),
 frozenset({'y'})]
```

In [ ]:

In [40]:
```
def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):
                if not can in ssCnt: ssCnt[can]=1
                else: ssCnt[can] += 1
    numItems = float(len(D))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key]/numItems
        if support >= minSupport:
            retList.insert(0,key)
        supportData[key] = support
    return retList, supportData
```

In [41]:
```
D = list(map(set,te))
D
```

Out[41]:
```
[{'_', 'a', 'c', 'd', 'e', 'o', 'r', 't'},
 {'_', 'a', 'd', 'e', 'i', 'l', 's'},
 {'_', 'a', 'c', 'd', 'e', 'i', 'n', 'o', 'p', 'r', 's', 'y'},
 {'_', 'a', 'd', 'e', 'i', 'm', 'n', 'p', 'r', 't'},
 {'_', 'a', 'e', 'l', 's', 't', 'v'},
 {'_', 'd', 'e', 'o', 'r', 'w'},
 {'_', 'a', 'd', 'e', 'f', 'h', 'o', 'r', 'u', 'y'},
 {'_', 'd', 'e', 'i', 'o', 'r'},
 {'_', 'b', 'd', 'e', 'm', 'n', 'o', 'r', 'u'},
 {'_', 'c', 'd', 'i', 'o', 'p', 'r', 't', 'u'},
 {'_', 'a', 'c', 'd', 'e', 'm', 'n', 'o', 'p', 'r', 't', 'u'},
 {'d', 'e', 'o', 'r'},
 {'_', 'd', 'e', 'i', 'r', 's', 'u'}]
```

In [42]:
```
L1,suppDat0 = scanD(D,C1,0.5)
L1
```

Out[42]:
```
[frozenset({'r'}),
 frozenset({'o'}),
 frozenset({'e'}),
 frozenset({'d'}),
 frozenset({'a'}),
 frozenset({'_'})]
```

In [43]:
```
def aprioriGen(Lk, k): #creates Ck
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]
            L1.sort(); L2.sort()
            if L1==L2: #if first k-2 elements are equal
                retList.append(Lk[i] | Lk[j]) #set union
    return retList
```

In [44]:
```
def apriori(te, minSupport = 0.5):
    C1 = createC1(te)
    D = list(map(set, te))
    L1, supportData = scanD(D, C1, minSupport)
    L = [L1]
    k = 2
    while (len(L[k-2]) > 0):
        Ck = aprioriGen(L[k-2], k)
        Lk, supK = scanD(D, Ck, minSupport)#scan DB to get Lk
        supportData.update(supK)
        L.append(Lk)
        k += 1
    return L, supportData
```

In [45]:
```
L,suppData = apriori(te)
```

In [46]:
```
L
```

Out[46]:
```
[[frozenset({'r'}),
  frozenset({'o'}),
  frozenset({'e'}),
  frozenset({'d'}),
  frozenset({'a'}),
  frozenset({'_'})],
 [frozenset({'_', 'a'}),
  frozenset({'_', 'd'}),
  frozenset({'_', 'e'}),
  frozenset({'a', 'e'}),
  frozenset({'d', 'e'}),
  frozenset({'_', 'o'}),
  frozenset({'d', 'o'}),
  frozenset({'e', 'o'}),
  frozenset({'_', 'r'}),
  frozenset({'d', 'r'}),
  frozenset({'e', 'r'}),
  frozenset({'o', 'r'})],
 [frozenset({'_', 'e', 'r'}),
  frozenset({'e', 'o', 'r'}),
  frozenset({'d', 'o', 'r'}),
  frozenset({'d', 'e', 'o'}),
  frozenset({'_', 'o', 'r'}),
  frozenset({'_', 'e', 'o'}),
  frozenset({'_', 'd', 'o'}),
  frozenset({'d', 'e', 'r'}),
  frozenset({'_', 'a', 'e'}),
  frozenset({'_', 'd', 'r'}),
  frozenset({'_', 'd', 'e'})],
 [frozenset({'_', 'd', 'e', 'r'}),
  frozenset({'_', 'd', 'e', 'o'}),
  frozenset({'_', 'd', 'o', 'r'}),
  frozenset({'d', 'e', 'o', 'r'}),
  frozenset({'_', 'e', 'o', 'r'})],
 [frozenset({'_', 'd', 'e', 'o', 'r'})],
 []]
```

In [47]:
```
aprioriGen(L[0],2)
```

Out[47]:
```
[frozenset({'o', 'r'}),
 frozenset({'e', 'r'}),
 frozenset({'d', 'r'}),
 frozenset({'a', 'r'}),
 frozenset({'_', 'r'}),
 frozenset({'e', 'o'}),
 frozenset({'d', 'o'}),
 frozenset({'a', 'o'}),
 frozenset({'_', 'o'}),
 frozenset({'d', 'e'}),
 frozenset({'a', 'e'}),
 frozenset({'_', 'e'}),
 frozenset({'a', 'd'}),
 frozenset({'_', 'd'}),
 frozenset({'_', 'a'})]
```

In [48]:
```
def generateRules(L, supportData, minConf=0.7):  #supportData is a dict coming
 from scanD
    bigRuleList = []
    for i in range(1, len(L)):#only get the sets with two or more items
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList,
minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList
```

In [49]:
```
def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = [] #create new list to return
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq] #calc confiden
ce
        if conf >= minConf:
            print (freqSet-conseq,'-->',conseq,'conf:',conf)
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH
```

In [50]:
```
def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])
    if (len(freqSet) > (m + 1)): #try further merging
        Hmp1 = aprioriGen(H, m+1)#create Hm+1 new candidates
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1):    #need at least two sets to merge
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)
```

In [51]:
```
L,suppData= apriori(te,minSupport=0.5)
```

In [53]:
```
L
```

Out[53]:
```
[[frozenset({'r'}),
  frozenset({'o'}),
  frozenset({'e'}),
  frozenset({'d'}),
  frozenset({'a'}),
  frozenset({'_'})],
 [frozenset({'_', 'a'}),
  frozenset({'_', 'd'}),
  frozenset({'_', 'e'}),
  frozenset({'a', 'e'}),
  frozenset({'d', 'e'}),
  frozenset({'_', 'o'}),
  frozenset({'d', 'o'}),
  frozenset({'e', 'o'}),
  frozenset({'_', 'r'}),
  frozenset({'d', 'r'}),
  frozenset({'e', 'r'}),
  frozenset({'o', 'r'})],
 [frozenset({'_', 'e', 'r'}),
  frozenset({'e', 'o', 'r'}),
  frozenset({'d', 'o', 'r'}),
  frozenset({'d', 'e', 'o'}),
  frozenset({'_', 'o', 'r'}),
  frozenset({'_', 'e', 'o'}),
  frozenset({'_', 'd', 'o'}),
  frozenset({'d', 'e', 'r'}),
  frozenset({'_', 'a', 'e'}),
  frozenset({'_', 'd', 'r'}),
  frozenset({'_', 'd', 'e'})],
 [frozenset({'_', 'd', 'e', 'r'}),
  frozenset({'_', 'd', 'e', 'o'}),
  frozenset({'_', 'd', 'o', 'r'}),
  frozenset({'d', 'e', 'o', 'r'}),
  frozenset({'_', 'e', 'o', 'r'})],
 [frozenset({'_', 'd', 'e', 'o', 'r'})],
 []]
```

In [52]:
```
rules= generateRules(L,suppData, minConf=0.7)
rules
```

```
frozenset({'a'}) --> frozenset({'_'}) conf: 1.0
frozenset({'_'}) --> frozenset({'d'}) conf: 0.9166666666666666
frozenset({'d'}) --> frozenset({'_'}) conf: 0.9166666666666666
frozenset({'_'}) --> frozenset({'e'}) conf: 0.9166666666666666
frozenset({'e'}) --> frozenset({'_'}) conf: 0.9166666666666666
frozenset({'a'}) --> frozenset({'e'}) conf: 1.0
frozenset({'e'}) --> frozenset({'d'}) conf: 0.9166666666666666
frozenset({'d'}) --> frozenset({'e'}) conf: 0.9166666666666666
frozenset({'o'}) --> frozenset({'_'}) conf: 0.888888888888889
frozenset({'d'}) --> frozenset({'o'}) conf: 0.7499999999999999
frozenset({'o'}) --> frozenset({'d'}) conf: 1.0
frozenset({'o'}) --> frozenset({'e'}) conf: 0.888888888888889
frozenset({'_'}) --> frozenset({'r'}) conf: 0.8333333333333334
frozenset({'r'}) --> frozenset({'_'}) conf: 0.9090909090909092
frozenset({'r'}) --> frozenset({'d'}) conf: 1.0
frozenset({'d'}) --> frozenset({'r'}) conf: 0.9166666666666666
frozenset({'e'}) --> frozenset({'r'}) conf: 0.8333333333333334
frozenset({'r'}) --> frozenset({'e'}) conf: 0.9090909090909092
frozenset({'r'}) --> frozenset({'o'}) conf: 0.8181818181818181
frozenset({'o'}) --> frozenset({'r'}) conf: 1.0
frozenset({'_'}) --> frozenset({'r', 'e'}) conf: 0.7499999999999999
frozenset({'e'}) --> frozenset({'r', '_'}) conf: 0.7499999999999999
frozenset({'r'}) --> frozenset({'e', '_'}) conf: 0.8181818181818181
frozenset({'r'}) --> frozenset({'o', 'e'}) conf: 0.7272727272727273
frozenset({'o'}) --> frozenset({'e', 'r'}) conf: 0.888888888888889
frozenset({'r'}) --> frozenset({'o', 'd'}) conf: 0.8181818181818181
frozenset({'d'}) --> frozenset({'o', 'r'}) conf: 0.7499999999999999
frozenset({'o'}) --> frozenset({'d', 'r'}) conf: 1.0
frozenset({'o'}) --> frozenset({'d', 'e'}) conf: 0.888888888888889
frozenset({'r'}) --> frozenset({'o', '_'}) conf: 0.7272727272727273
frozenset({'o'}) --> frozenset({'r', '_'}) conf: 0.888888888888889
frozenset({'o'}) --> frozenset({'e', '_'}) conf: 0.7777777777777778
frozenset({'o'}) --> frozenset({'d', '_'}) conf: 0.888888888888889
frozenset({'r'}) --> frozenset({'d', 'e'}) conf: 0.9090909090909092
frozenset({'e'}) --> frozenset({'d', 'r'}) conf: 0.8333333333333334
frozenset({'d'}) --> frozenset({'e', 'r'}) conf: 0.8333333333333334
frozenset({'a'}) --> frozenset({'e', '_'}) conf: 1.0
frozenset({'_'}) --> frozenset({'d', 'r'}) conf: 0.8333333333333334
frozenset({'r'}) --> frozenset({'d', '_'}) conf: 0.9090909090909092
frozenset({'d'}) --> frozenset({'r', '_'}) conf: 0.8333333333333334
frozenset({'_'}) --> frozenset({'d', 'e'}) conf: 0.8333333333333334
frozenset({'e'}) --> frozenset({'d', '_'}) conf: 0.8333333333333334
frozenset({'d'}) --> frozenset({'e', '_'}) conf: 0.8333333333333334
frozenset({'e', '_'}) --> frozenset({'d', 'r'}) conf: 0.8181818181818181
frozenset({'r', '_'}) --> frozenset({'d', 'e'}) conf: 0.8999999999999999
frozenset({'r', 'e'}) --> frozenset({'d', '_'}) conf: 0.8999999999999999
frozenset({'d', '_'}) --> frozenset({'r', 'e'}) conf: 0.8181818181818181
frozenset({'d', 'e'}) --> frozenset({'r', '_'}) conf: 0.8181818181818181
frozenset({'d', 'r'}) --> frozenset({'e', '_'}) conf: 0.8181818181818181
frozenset({'_'}) --> frozenset({'d', 'r', 'e'}) conf: 0.7499999999999999
frozenset({'e'}) --> frozenset({'d', 'r', '_'}) conf: 0.7499999999999999
frozenset({'r'}) --> frozenset({'d', 'e', '_'}) conf: 0.8181818181818181
frozenset({'d'}) --> frozenset({'r', 'e', '_'}) conf: 0.7499999999999999
frozenset({'o', '_'}) --> frozenset({'d', 'e'}) conf: 0.8749999999999999
frozenset({'o', 'e'}) --> frozenset({'d', '_'}) conf: 0.8749999999999999
frozenset({'o', 'd'}) --> frozenset({'e', '_'}) conf: 0.7777777777777778
frozenset({'o'}) --> frozenset({'d', 'e', '_'}) conf: 0.7777777777777778
```

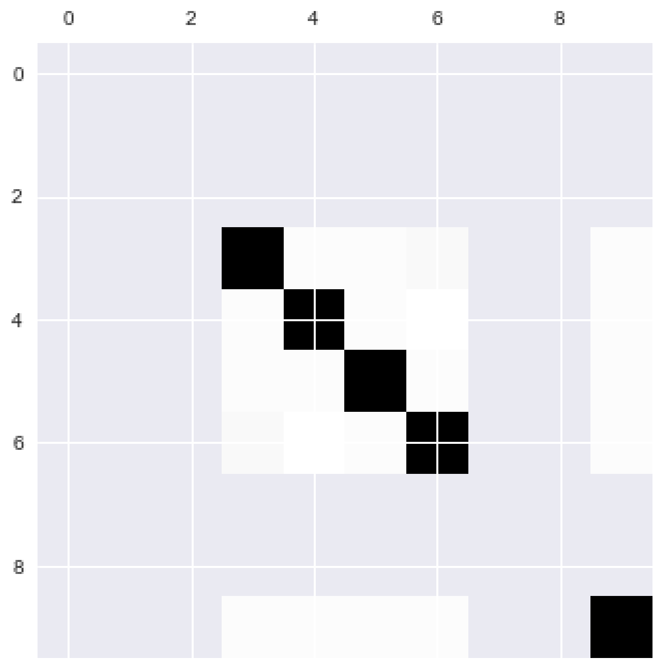
```
frozenset({'r', '_'}) --> frozenset({'o', 'd'}) conf: 0.8
frozenset({'o', '_'}) --> frozenset({'d', 'r'}) conf: 1.0
frozenset({'o', 'r'}) --> frozenset({'d', '_'}) conf: 0.888888888888889
frozenset({'d', '_'}) --> frozenset({'o', 'r'}) conf: 0.7272727272727273
frozenset({'d', 'r'}) --> frozenset({'o', '_'}) conf: 0.7272727272727273
frozenset({'o', 'd'}) --> frozenset({'r', '_'}) conf: 0.888888888888889
frozenset({'r'}) --> frozenset({'o', 'd', '_'}) conf: 0.7272727272727273
frozenset({'o'}) --> frozenset({'d', 'r', '_'}) conf: 0.888888888888889
frozenset({'r', 'e'}) --> frozenset({'o', 'd'}) conf: 0.8
frozenset({'o', 'e'}) --> frozenset({'d', 'r'}) conf: 1.0
frozenset({'o', 'r'}) --> frozenset({'d', 'e'}) conf: 0.888888888888889
frozenset({'d', 'e'}) --> frozenset({'o', 'r'}) conf: 0.7272727272727273
frozenset({'d', 'r'}) --> frozenset({'o', 'e'}) conf: 0.7272727272727273
frozenset({'o', 'd'}) --> frozenset({'r', 'e'}) conf: 0.888888888888889
frozenset({'r'}) --> frozenset({'o', 'd', 'e'}) conf: 0.7272727272727273
frozenset({'o'}) --> frozenset({'d', 'r', 'e'}) conf: 0.888888888888889
frozenset({'r', '_'}) --> frozenset({'o', 'e'}) conf: 0.7
frozenset({'r', 'e'}) --> frozenset({'o', '_'}) conf: 0.7
frozenset({'o', '_'}) --> frozenset({'r', 'e'}) conf: 0.8749999999999999
frozenset({'o', 'e'}) --> frozenset({'r', '_'}) conf: 0.8749999999999999
frozenset({'o', 'r'}) --> frozenset({'e', '_'}) conf: 0.7777777777777778
frozenset({'o'}) --> frozenset({'r', 'e', '_'}) conf: 0.7777777777777778
frozenset({'o', 'r', '_'}) --> frozenset({'d', 'e'}) conf: 0.8749999999999999
frozenset({'d', 'r', '_'}) --> frozenset({'o', 'e'}) conf: 0.7
frozenset({'o', 'd', '_'}) --> frozenset({'e', 'r'}) conf: 0.8749999999999999
frozenset({'o', 'd', 'r'}) --> frozenset({'e', '_'}) conf: 0.7777777777777778
frozenset({'e', 'r', '_'}) --> frozenset({'o', 'd'}) conf: 0.7777777777777778
frozenset({'o', 'e', '_'}) --> frozenset({'d', 'r'}) conf: 1.0
frozenset({'o', 'e', 'r'}) --> frozenset({'d', '_'}) conf: 0.8749999999999999
frozenset({'d', 'e', '_'}) --> frozenset({'o', 'r'}) conf: 0.7
frozenset({'d', 'e', 'r'}) --> frozenset({'o', '_'}) conf: 0.7
frozenset({'d', 'e', 'o'}) --> frozenset({'r', '_'}) conf: 0.8749999999999999
frozenset({'o', '_'}) --> frozenset({'d', 'e', 'r'}) conf: 0.8749999999999999
frozenset({'o', 'r'}) --> frozenset({'d', 'e', '_'}) conf: 0.7777777777777778
frozenset({'r', '_'}) --> frozenset({'o', 'e', 'd'}) conf: 0.7
frozenset({'o', 'd'}) --> frozenset({'e', 'r', '_'}) conf: 0.7777777777777778
frozenset({'e', 'r'}) --> frozenset({'o', 'd', '_'}) conf: 0.7
frozenset({'o', 'e'}) --> frozenset({'d', 'r', '_'}) conf: 0.8749999999999999
frozenset({'o'}) --> frozenset({'d', 'r', 'e', '_'}) conf: 0.7777777777777778
```

```
Out[52]: [(frozenset({'a'}), frozenset({'_'}), 1.0),
          (frozenset({'_'}), frozenset({'d'}), 0.9166666666666666),
          (frozenset({'d'}), frozenset({'_'}), 0.9166666666666666),
          (frozenset({'_'}), frozenset({'e'}), 0.9166666666666666),
          (frozenset({'e'}), frozenset({'_'}), 0.9166666666666666),
          (frozenset({'a'}), frozenset({'e'}), 1.0),
          (frozenset({'e'}), frozenset({'d'}), 0.9166666666666666),
          (frozenset({'d'}), frozenset({'e'}), 0.9166666666666666),
          (frozenset({'o'}), frozenset({'_'}), 0.888888888888889),
          (frozenset({'d'}), frozenset({'o'}), 0.7499999999999999),
          (frozenset({'o'}), frozenset({'d'}), 1.0),
          (frozenset({'o'}), frozenset({'e'}), 0.888888888888889),
          (frozenset({'_'}), frozenset({'r'}), 0.8333333333333334),
          (frozenset({'r'}), frozenset({'_'}), 0.9090909090909092),
          (frozenset({'r'}), frozenset({'d'}), 1.0),
          (frozenset({'d'}), frozenset({'r'}), 0.9166666666666666),
          (frozenset({'e'}), frozenset({'r'}), 0.8333333333333334),
          (frozenset({'r'}), frozenset({'e'}), 0.9090909090909092),
          (frozenset({'r'}), frozenset({'o'}), 0.8181818181818181),
          (frozenset({'o'}), frozenset({'r'}), 1.0),
          (frozenset({'_'}), frozenset({'e', 'r'}), 0.7499999999999999),
          (frozenset({'e'}), frozenset({'_', 'r'}), 0.7499999999999999),
          (frozenset({'r'}), frozenset({'_', 'e'}), 0.8181818181818181),
          (frozenset({'r'}), frozenset({'e', 'o'}), 0.7272727272727273),
          (frozenset({'o'}), frozenset({'e', 'r'}), 0.888888888888889),
          (frozenset({'r'}), frozenset({'d', 'o'}), 0.8181818181818181),
          (frozenset({'d'}), frozenset({'o', 'r'}), 0.7499999999999999),
          (frozenset({'o'}), frozenset({'d', 'r'}), 1.0),
          (frozenset({'o'}), frozenset({'d', 'e'}), 0.888888888888889),
          (frozenset({'r'}), frozenset({'_', 'o'}), 0.7272727272727273),
          (frozenset({'o'}), frozenset({'_', 'r'}), 0.888888888888889),
          (frozenset({'o'}), frozenset({'_', 'e'}), 0.7777777777777778),
          (frozenset({'o'}), frozenset({'_', 'd'}), 0.888888888888889),
          (frozenset({'r'}), frozenset({'d', 'e'}), 0.9090909090909092),
          (frozenset({'e'}), frozenset({'d', 'r'}), 0.8333333333333334),
          (frozenset({'d'}), frozenset({'e', 'r'}), 0.8333333333333334),
          (frozenset({'a'}), frozenset({'_', 'e'}), 1.0),
          (frozenset({'_'}), frozenset({'d', 'r'}), 0.8333333333333334),
          (frozenset({'r'}), frozenset({'_', 'd'}), 0.9090909090909092),
          (frozenset({'d'}), frozenset({'_', 'r'}), 0.8333333333333334),
          (frozenset({'_'}), frozenset({'d', 'e'}), 0.8333333333333334),
          (frozenset({'e'}), frozenset({'_', 'd'}), 0.8333333333333334),
          (frozenset({'d'}), frozenset({'_', 'e'}), 0.8333333333333334),
          (frozenset({'_', 'e'}), frozenset({'d', 'r'}), 0.8181818181818181),
          (frozenset({'_', 'r'}), frozenset({'d', 'e'}), 0.8999999999999999),
          (frozenset({'e', 'r'}), frozenset({'_', 'd'}), 0.8999999999999999),
          (frozenset({'_', 'd'}), frozenset({'e', 'r'}), 0.8181818181818181),
          (frozenset({'d', 'e'}), frozenset({'_', 'r'}), 0.8181818181818181),
          (frozenset({'d', 'r'}), frozenset({'_', 'e'}), 0.8181818181818181),
          (frozenset({'_'}), frozenset({'d', 'e', 'r'}), 0.7499999999999999),
          (frozenset({'e'}), frozenset({'_', 'd', 'r'}), 0.7499999999999999),
          (frozenset({'r'}), frozenset({'_', 'd', 'e'}), 0.8181818181818181),
          (frozenset({'d'}), frozenset({'_', 'e', 'r'}), 0.7499999999999999),
          (frozenset({'_', 'o'}), frozenset({'d', 'e'}), 0.8749999999999999),
          (frozenset({'e', 'o'}), frozenset({'_', 'd'}), 0.8749999999999999),
          (frozenset({'d', 'o'}), frozenset({'_', 'e'}), 0.7777777777777778),
          (frozenset({'o'}), frozenset({'_', 'd', 'e'}), 0.7777777777777778),
```

```
(frozenset({'_', 'r'}), frozenset({'d', 'o'}), 0.8),
(frozenset({'_', 'o'}), frozenset({'d', 'r'}), 1.0),
(frozenset({'o', 'r'}), frozenset({'_', 'd'}), 0.888888888888889),
(frozenset({'_', 'd'}), frozenset({'o', 'r'}), 0.7272727272727273),
(frozenset({'d', 'r'}), frozenset({'_', 'o'}), 0.7272727272727273),
(frozenset({'d', 'o'}), frozenset({'_', 'r'}), 0.888888888888889),
(frozenset({'r'}), frozenset({'_', 'd', 'o'}), 0.7272727272727273),
(frozenset({'o'}), frozenset({'_', 'd', 'r'}), 0.888888888888889),
(frozenset({'e', 'r'}), frozenset({'d', 'o'}), 0.8),
(frozenset({'e', 'o'}), frozenset({'d', 'r'}), 1.0),
(frozenset({'o', 'r'}), frozenset({'d', 'e'}), 0.888888888888889),
(frozenset({'d', 'e'}), frozenset({'o', 'r'}), 0.7272727272727273),
(frozenset({'d', 'r'}), frozenset({'e', 'o'}), 0.7272727272727273),
(frozenset({'d', 'o'}), frozenset({'e', 'r'}), 0.888888888888889),
(frozenset({'r'}), frozenset({'d', 'e', 'o'}), 0.7272727272727273),
(frozenset({'o'}), frozenset({'d', 'e', 'r'}), 0.888888888888889),
(frozenset({'_', 'r'}), frozenset({'e', 'o'}), 0.7),
(frozenset({'e', 'r'}), frozenset({'_', 'o'}), 0.7),
(frozenset({'_', 'o'}), frozenset({'e', 'r'}), 0.8749999999999999),
(frozenset({'e', 'o'}), frozenset({'_', 'r'}), 0.8749999999999999),
(frozenset({'o', 'r'}), frozenset({'_', 'e'}), 0.7777777777777778),
(frozenset({'o'}), frozenset({'_', 'e', 'r'}), 0.7777777777777778),
(frozenset({'_', 'o', 'r'}), frozenset({'d', 'e'}), 0.8749999999999999),
(frozenset({'_', 'd', 'r'}), frozenset({'e', 'o'}), 0.7),
(frozenset({'_', 'd', 'o'}), frozenset({'e', 'r'}), 0.8749999999999999),
(frozenset({'d', 'o', 'r'}), frozenset({'_', 'e'}), 0.7777777777777778),
(frozenset({'_', 'e', 'r'}), frozenset({'d', 'o'}), 0.7777777777777778),
(frozenset({'_', 'e', 'o'}), frozenset({'d', 'r'}), 1.0),
(frozenset({'e', 'o', 'r'}), frozenset({'_', 'd'}), 0.8749999999999999),
(frozenset({'_', 'd', 'e'}), frozenset({'o', 'r'}), 0.7),
(frozenset({'d', 'e', 'r'}), frozenset({'_', 'o'}), 0.7),
(frozenset({'d', 'e', 'o'}), frozenset({'_', 'r'}), 0.8749999999999999),
(frozenset({'_', 'o'}), frozenset({'d', 'e', 'r'}), 0.8749999999999999),
(frozenset({'o', 'r'}), frozenset({'_', 'd', 'e'}), 0.7777777777777778),
(frozenset({'_', 'r'}), frozenset({'d', 'e', 'o'}), 0.7),
(frozenset({'d', 'o'}), frozenset({'_', 'e', 'r'}), 0.7777777777777778),
(frozenset({'e', 'r'}), frozenset({'_', 'd', 'o'}), 0.7),
(frozenset({'e', 'o'}), frozenset({'_', 'd', 'r'}), 0.8749999999999999),
(frozenset({'o'}), frozenset({'_', 'd', 'e', 'r'}), 0.7777777777777778)]
```

In [55]:
```
plt.matshow(te.corr())
```

Out[55]:
```
<matplotlib.image.AxesImage at 0x1478b6fd978>
```



In [44]:
```
conclusion
100 % frequent and co-occurring associations among a collection of products in
 the used data Instacart data set
```

In [22]:
```
{frozen} , {other} confidence = 1.0
{other} , {international} confidence = 1.0
{pantry}, {alcohol} confidence=1.0
{canned goods}, {alcohol} confidence=1.0
{pantry}, {canned goods} confidence=1.0
```

In [17]:
```

```

In [19]:
```

```