Capstone Project 2 - Final Report

Instacart - what's in the next order?

Background

In the era of online shopping, we can buy almost anything in the span of a click - from electronics to clothing and groceries is also added to the list. Companies like Instacart are grabbing much clientele, helping the customers with grocery shopping needs with the convenience of being able to place an order from anywhere.

Instacart is a technology company which provides same day grocery delivery and pick-up service in the US and Canada. Customers shop for groceries through the Instacart mobile app or Instacart.com from the company's more than 300 national, regional and local retailer partners. The order is then shopped and delivered by an Instacart personal shopper.

Instacart's customers can access the service via the Instacart mobile app or website and begin shopping by selecting their city and store, then adding groceries to their digital cart. Customers can pay with personal debit or credit cards, Android Pay and Apple Pay.

Problem Description

Physical stores aid in visually introducing other items to the shopper which might potentially become an item in the purchase. In comparison with brick and mortar stores, the online stores give less exposure to other products (smaller visual field) and therefore the choice of product display on the page is of critical importance to ensure a fulfilling shopping experience.

Online grocery shopping, although being a great convenience, poses a particular problem - stickiness and customer retention. Creating a good shopping experience with increasing sales and seamless experience is becoming increasingly difficult for e-commerce websites. As a part of enhancing user experience, the project tries to predict which previously purchased products will be in a user's next order. This is done by splitting the general reordering (or not) prediction and no-reorder prediction and combining the above models to arrive at a more generalized model taking into

consideration the User as well as Product features and combining both in the final model.

Stakeholders and Impact

Instacart's ability to predict customer's buy and be able to recommend products based on buying pattern and adaptation to their changing needs expectations - improves stickiness, helps in creating a seamless experience to and eventually promote user base for Instacart and maximize revenue.

Reordering prediction and thereby making recommendations to customer will help in making the shopping experience very personalized and will help users in having a smooth and seamless shopping experience. It will also help in faster checkout in comparison with having to search through the catalog of products offered on the app/website.

In addition to the above, the above prediction (and recommendation to customers) will help retail partners to have increased and continued sales through Instacart.

Data

Recently, the company (Instacart) made a portion of the data public, giving access to several files (orders, products, aisles, departments) in .csv format.

Original data is available on Instacart website.

The dataset contains data on 3.4 million orders from 206,000 users involving 50,000 products.

Following data available in the form of .csv files :

1. 'Aisles.csv'

   Includes the aisle id and aisle name

	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

2. ‘Departments.csv’

   Includes department id and department name

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

3. ‘Order_products__prior.csv’

Includes order_id, product_id, sequence of adding to the order (item’s sequence in the order), reordered (1,0) indicates if the product is reordered

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

4. ‘Order_products__train.csv’

Similar to above where the dataframe specifies which products were purchased in each order.

	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

5. ‘Orders.csv’

   This contains the eval_set specified (prior,test,train) that an order belongs to.

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	NaN
1	2398795	1	prior	2	3	7	15.0
2	473747	1	prior	3	3	12	21.0
3	2254736	1	prior	4	4	7	29.0
4	431534	1	prior	5	4	15	28.0

6. ‘Products.csv’

   Contains list of products with their respective aisle_id and department_id

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

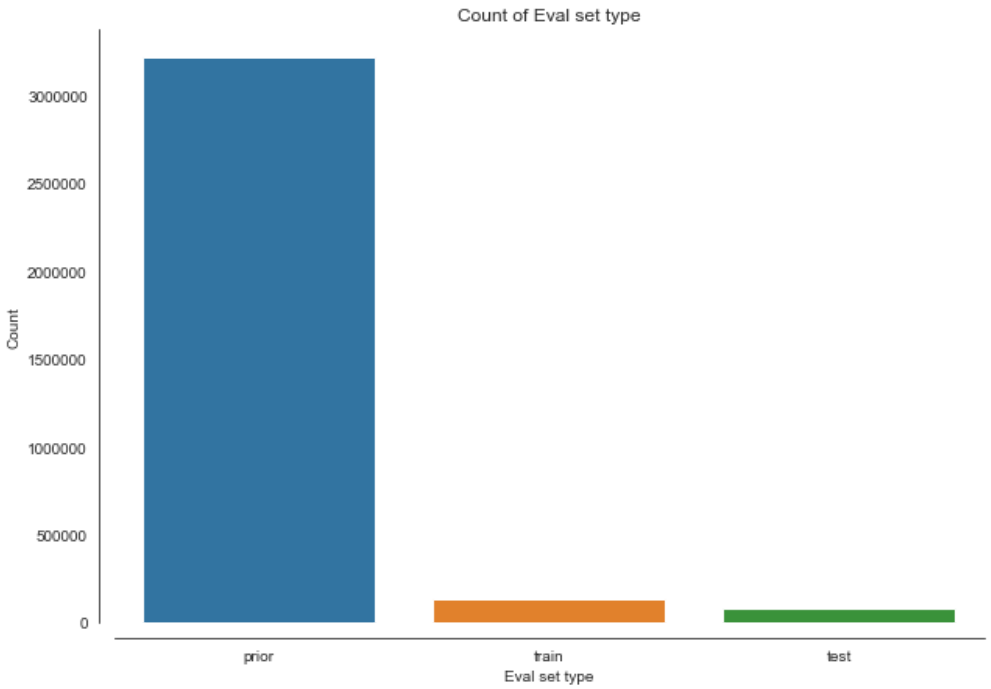
EXPLORATORY DATA ANALYSIS

With the available data, following EDA was carried out :

- Which day has the most orders?
- What time of the day are most orders placed?
- Lull period (between reorders) - frequency of reorder
- Which department do most reorders belong to?
- Which aisle do most reorders belong to ?
- Average number of items in a reorder
- Number of organic products in reorders
- Most reordered products

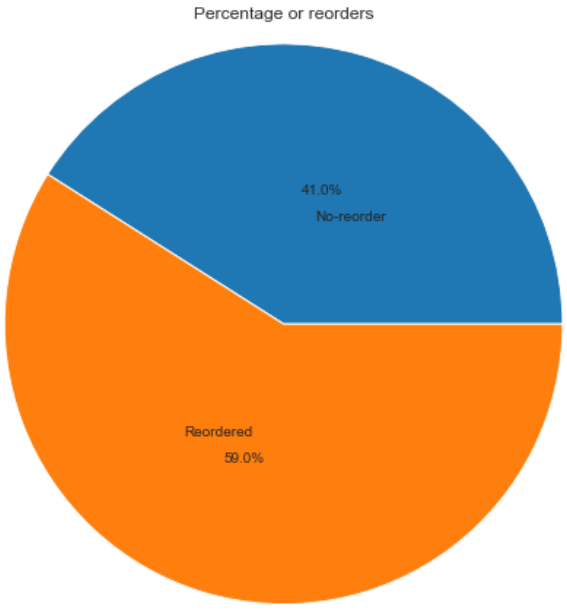
The orders dataset has 3241083 unique orders.
This dataframe has ‘days_since_prior_order’ column with NULL values.

```
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
```

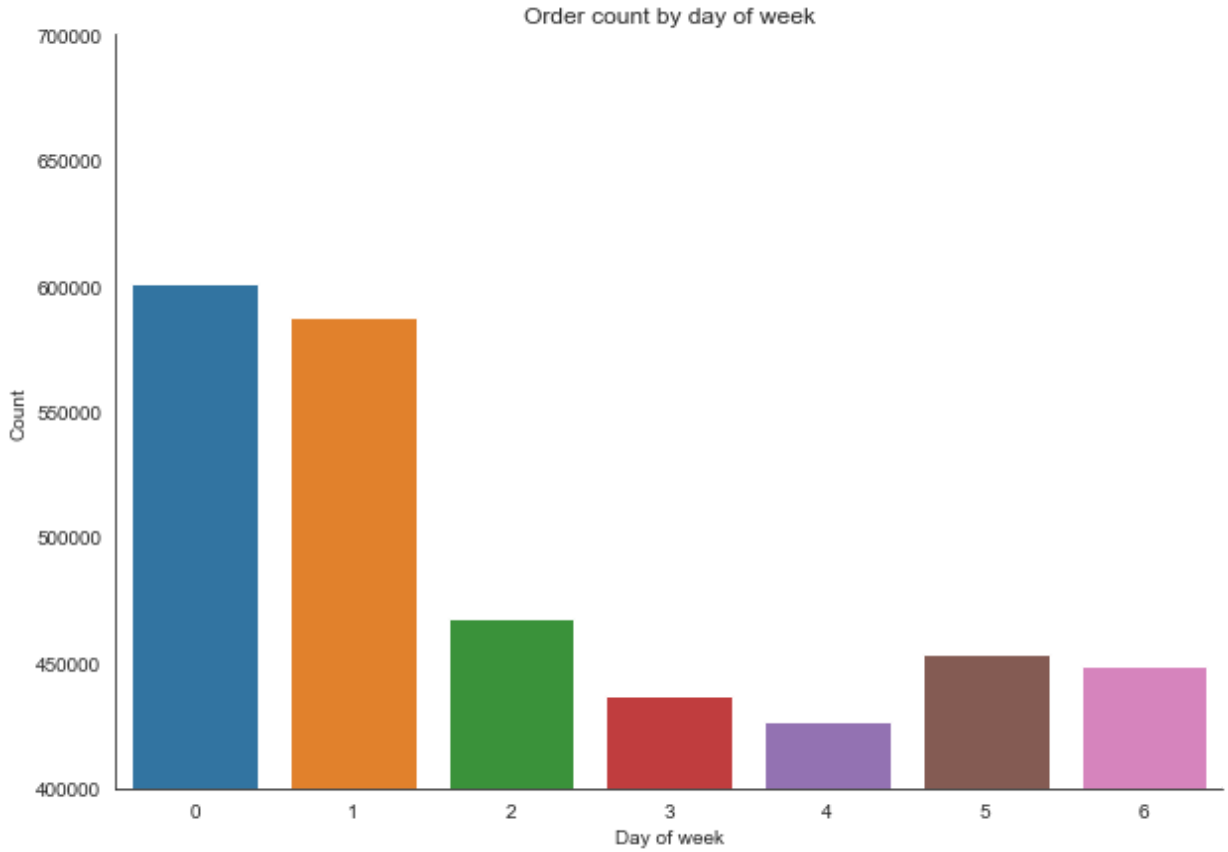


The above bar graph shows the classification of orders dataset (if it belongs to prior, train or test).

With the above dataframe, the number of reorders were calculated and observed that 41% of the rows - there were no reorders and rest 59% the entries were reorders.



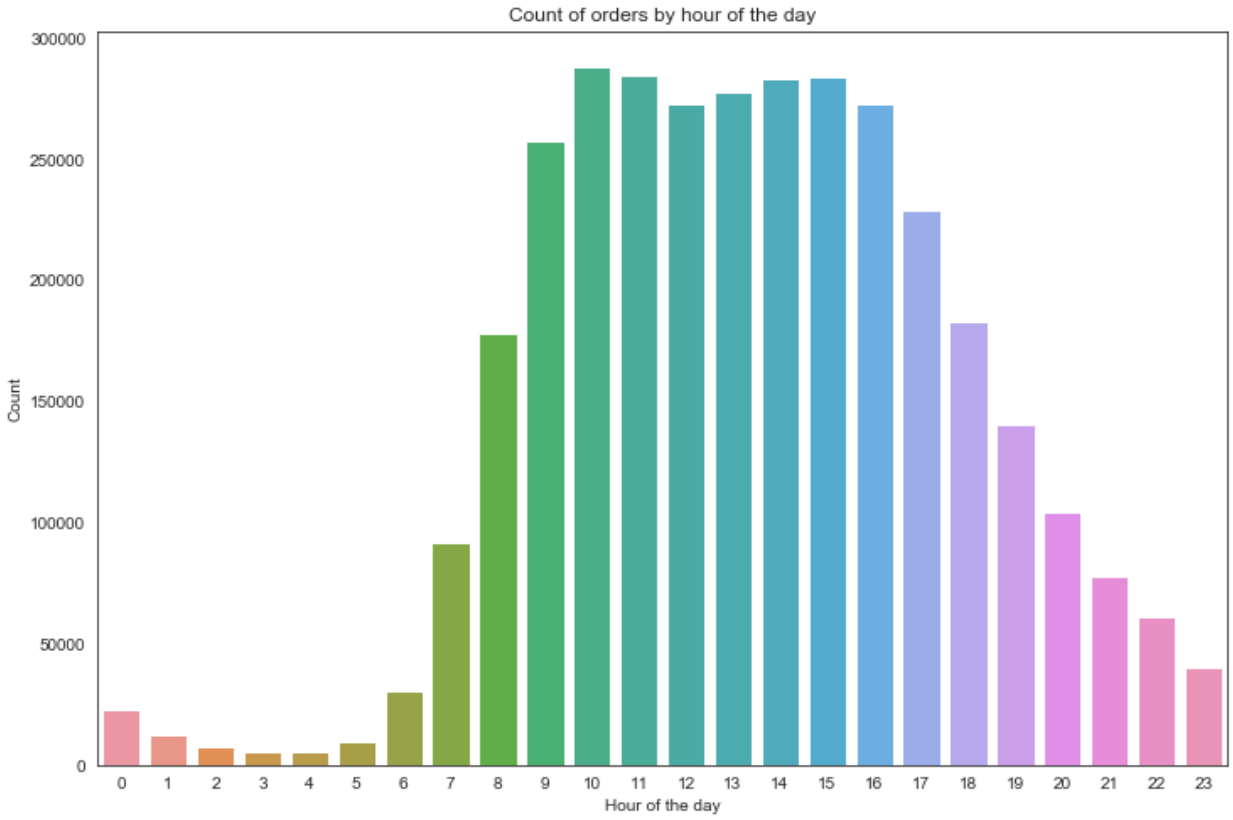
The orders dataframe was explored to understand the transaction trends. Basing on the day of the week, a histogram plot was drawn to see how the number of orders differed depending on the day of the week.



Order_dow column has no description of day of week and associated numerals - it is assumed that 0 corresponds to Saturday and 1 corresponds to Sunday (intuitively, most orders are placed on these two days)

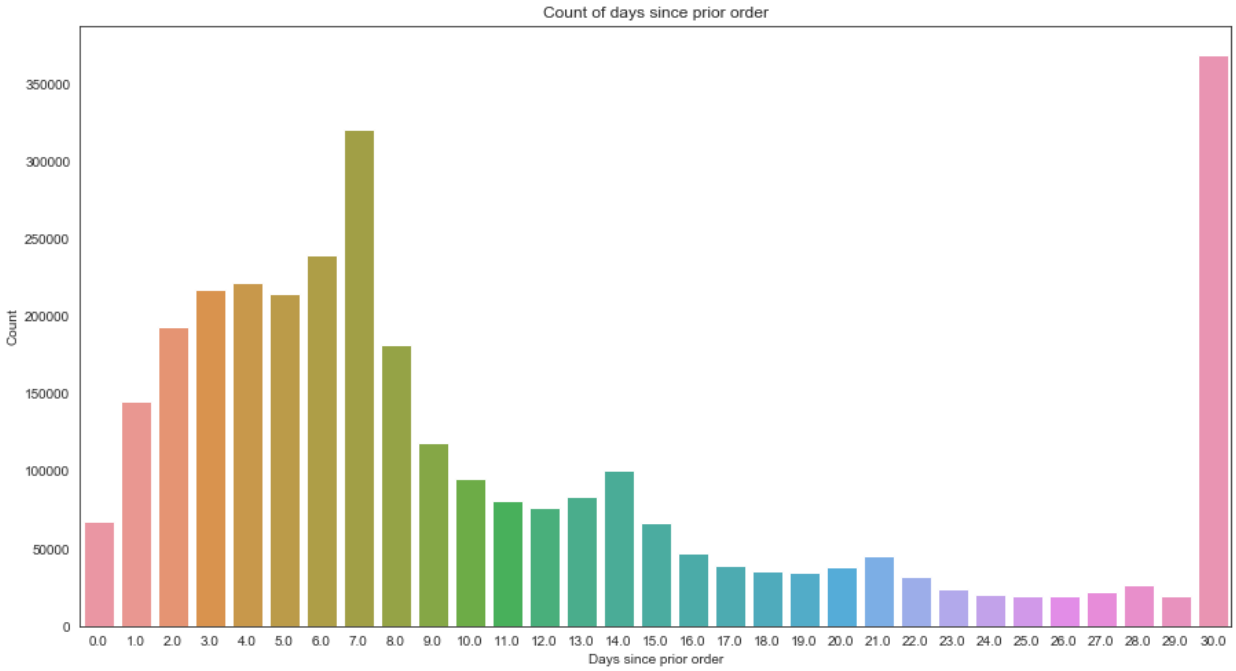
Above plot shows that most orders (600000 orders on Saturday and about 575000 orders on Sunday) are on the weekends in comparison with weekday orders.

To explore this further, the order_hour_of_day values were plotted to show how the orders were distributed by the hour of the day.



It is evident from the bar graph above that most orders are placed between 8 am and 8 pm. The least number of orders are placed between midnight and 6 am.

Weekend orders and order during daytime (8am to 8pm) were the trends observed using the orders dataframe. To check the frequency of reordering, we plotted the ‘days_since_prior_order’. We found some interesting trends in the below bar graph.

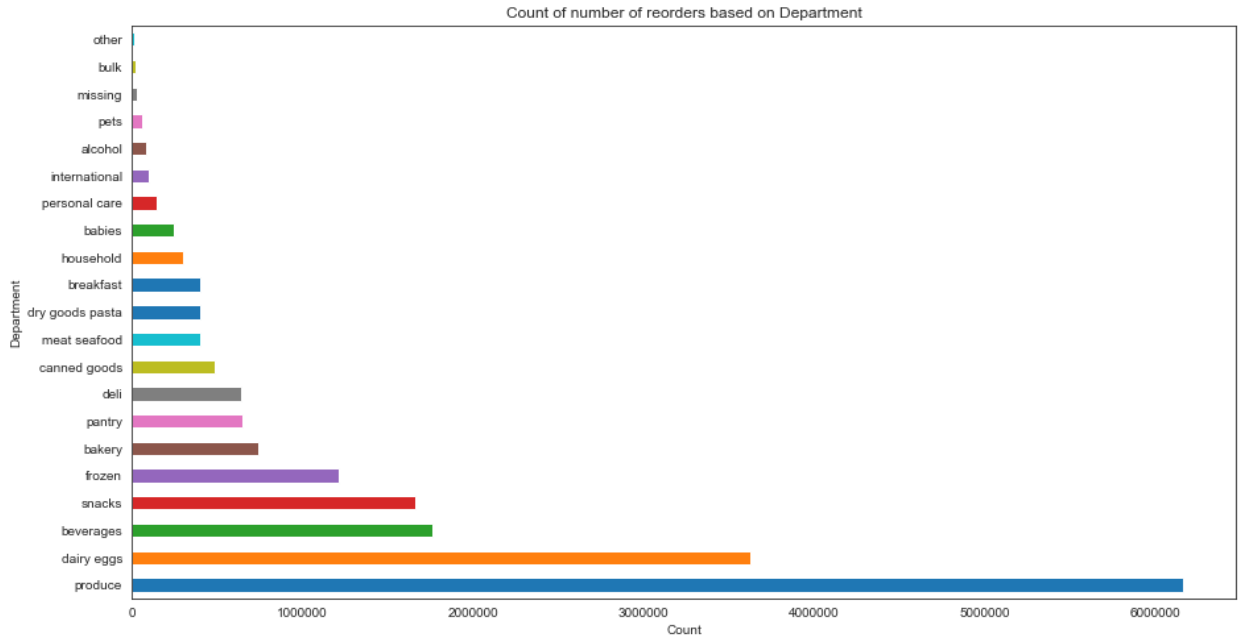


Above graph shows interesting trends - although there are several orders within 4-7 days since the last order, a significant number of orders are placed after 30 days of previous order.

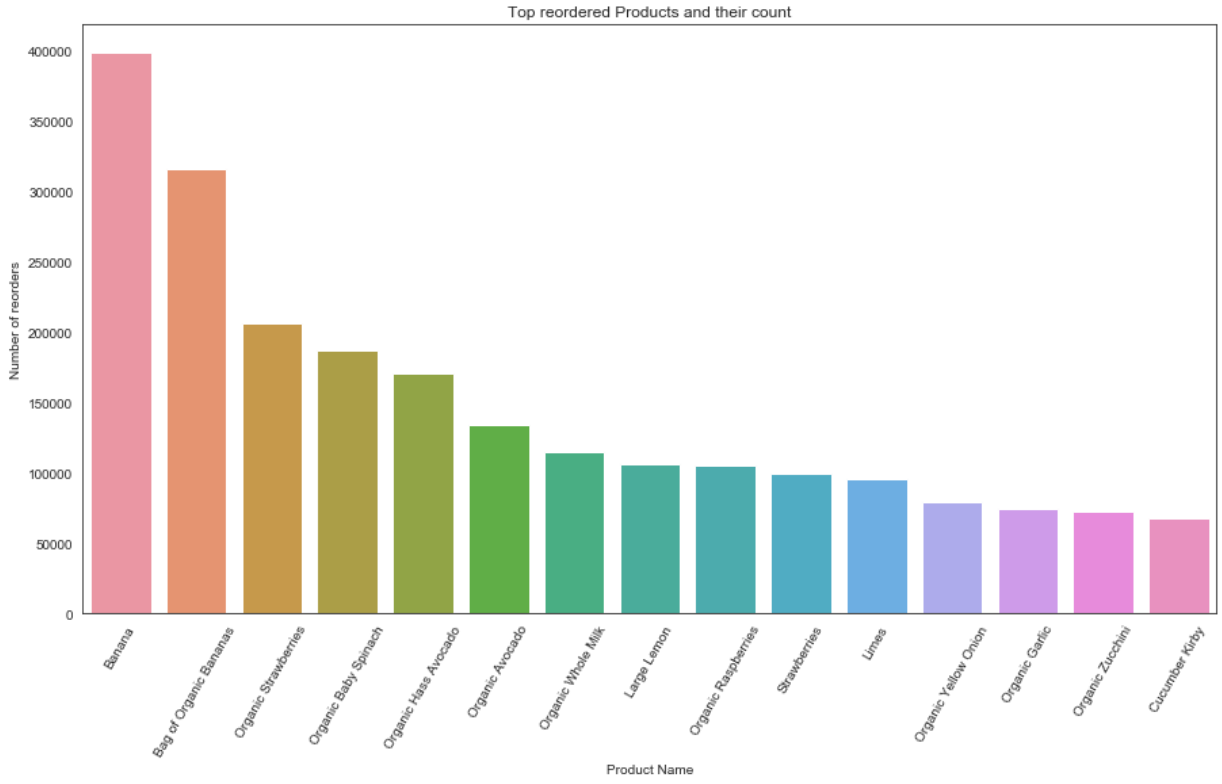
This trend is interesting as the number of next orders decrease after the 7 day period until the 30th day. Also, above spikes in reorder on specific days could possibly be because of a subscription service (reordering an item at user defined frequency) - although such details are not available in the dataset. Also, most reorders are observed between 3-7 days and 30th day which gives an indication that majority of the orders might be of fresh produce/poultry/dairy which has a shorter shelf life. The 30 day reordering frequency intuitively applies to other consumer goods which require monthly replenishment (general cleaning goods, paper products, etc).

EXPLORING REORDERED PRODUCTS

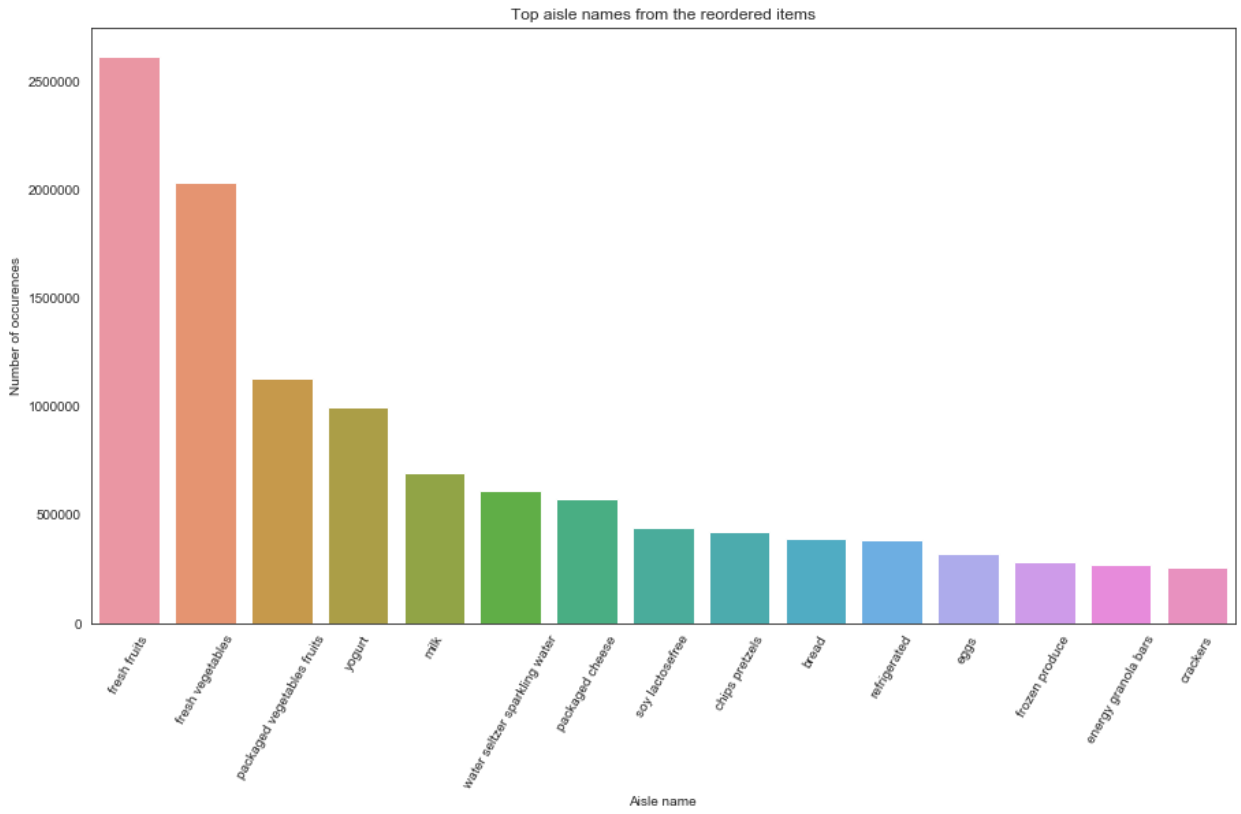
To confirm this trend, we filtered out dataset to show only reordered items (reordered = 1) and the number of reorders based on department was plotted as bar graphs and as assumed, produce has the maximum number of reorders, followed by dairy and beverages.



To explore the products which were reordered the most, we filtered our dataset to show reordered products by the product name.



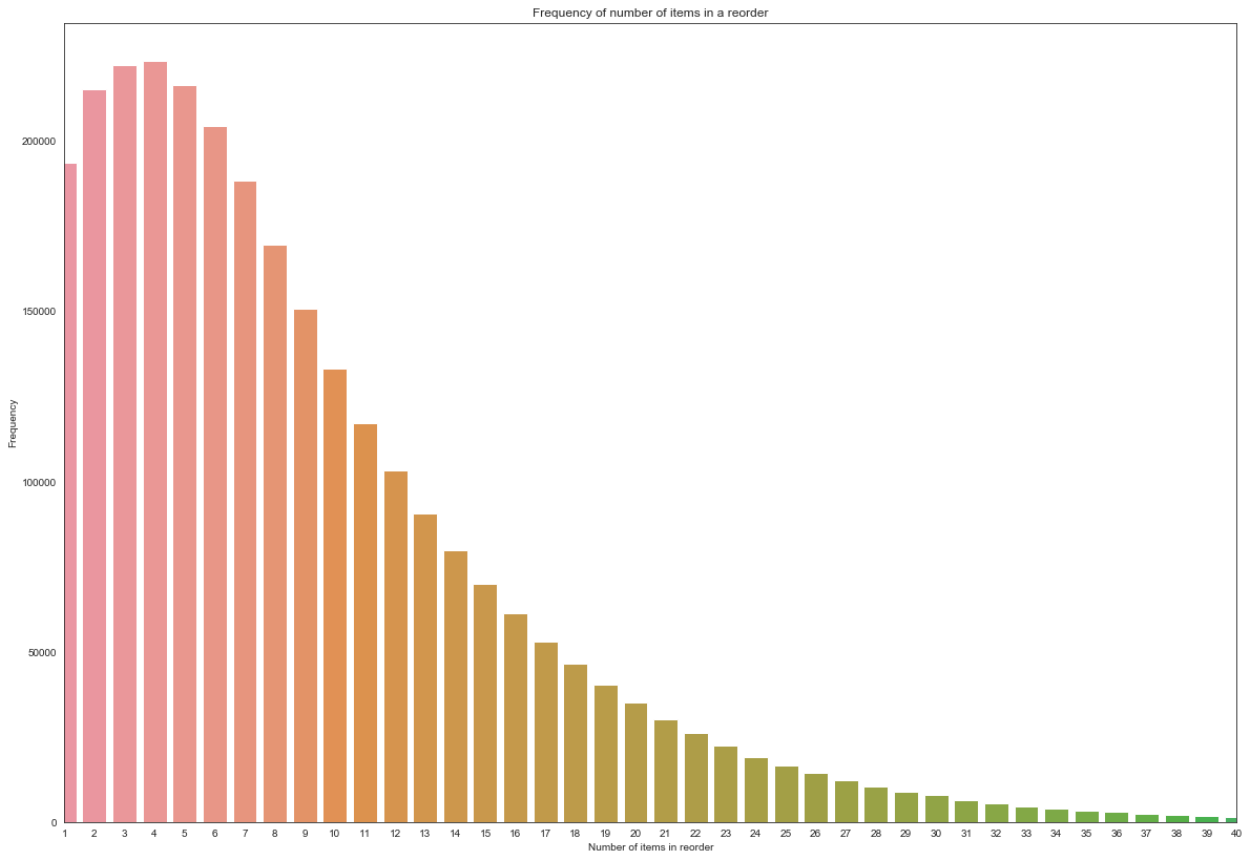
Top reordered products (which are from the produce department) - Banana, strawberries and Spinach top the list. Classifying the reorders based on the aisle names, we have fresh fruits aisle with maximum number of orders, followed by fresh vegetables, packaged vegetables/fruits and yogurt.



The above results confirm our assumption that most reorders between 4-7 days belong to produce and/or dairy products.

NUMBER OF PRODUCTS IN A REORDER

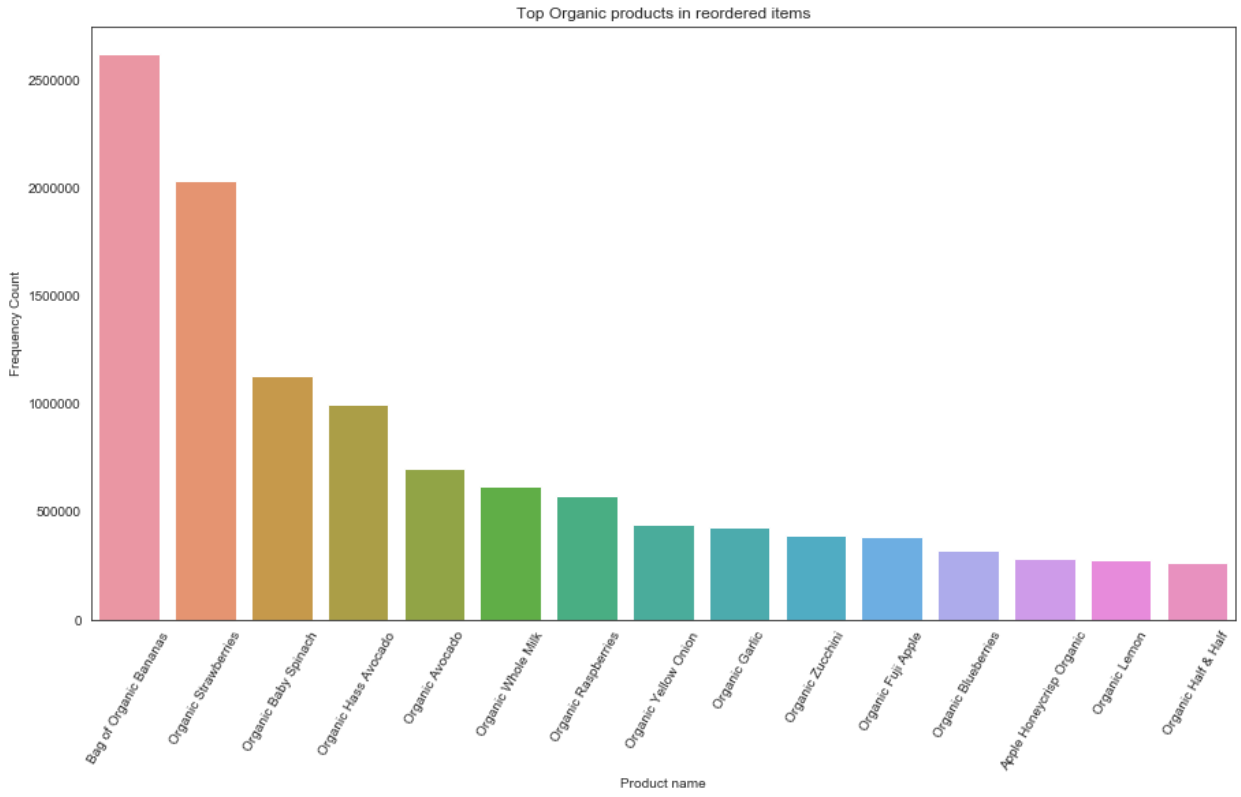
As most orders (or reorders) usually have more than one item, we further explore how many products are usually added to the cart in each order. A bar graph is drawn showing the number of products in each order and the count of number of orders.



From the above bar graph, most reorders have between 1 and 8 products.
Maximum number of reorders were with 4 products in the order.
Note : The number of items were limited to 40 for better visualization of graph.

ORGANIC PRODUCTS IN REORDERED ITEMS

To explore product names in the Organic section, we filtered the product name having ‘Organic’ and plotted a bar graph showing the most reordered organic products.



Bananas, strawberries, spinach, Avocado top the list of top organic items in a reorder.

User X Product Level Insights

To explore the User level and product level insights, we generate several new columns which are derived from the existing columns and use them as features in our model. To start with, we find out the

- Reorder ratio by department which is grouping the products by department and taking the mean of ‘reordered’ value of each product.

	department	reorder_ratio
7	dairy eggs	0.669969
3	beverages	0.653460
19	produce	0.649913
2	bakery	0.628141
8	deli	0.607719
18	pets	0.601285
1	babies	0.578971
5	bulk	0.577040
20	snacks	0.574180
0	alcohol	0.569924
13	meat seafood	0.567674
4	breakfast	0.560922
10	frozen	0.541885
9	dry goods pasta	0.461076
6	canned goods	0.457405
15	other	0.407980
11	household	0.402178
14	missing	0.395849
12	international	0.369229
16	pantry	0.346721
17	personal care	0.321129

Similarly reorder ratio by aisle is calculated by grouping products by aisle and taking the mean of reordered value of each product (belonging to that aisle)

	aisle	reorder_ratio
83	milk	0.781428
131	water seltzer sparkling water	0.729593
50	fresh fruits	0.718104
41	eggs	0.705366
119	soy lactosefree	0.692551
96	packaged produce	0.690734
133	yogurt	0.686489
33	cream	0.685046
11	bread	0.670168
110	refrigerated	0.663302
12	breakfast bakery	0.651222
43	energy sports drinks	0.649607
117	soft drinks	0.638832
98	packaged vegetables fruits	0.638514
132	white wines	0.630081
56	frozen breakfast	0.626221
23	cat food care	0.620883
14	bulk dried fruits vegetables	0.620394
128	trail mix snack mix	0.620230

Frequency of ordering of top reordered products

User level and Product Level details (to arrive at features necessary for our model)

The following features are extracted from the available dataframe :

USER LEVEL

- Maximum number of orders per user is determined
- With the above, reorder ratio of each User is calculated (by taking the mean of reordered values)
- Merge the above dataframes (inner join) to get reorder statistics for each user. We call it **W**

PRODUCT LEVEL

Number of reorders (purchase count) per product
Product reorder probability (mean of reordered value of each product)

Merging above dataframes (inner join) to get reordering statistics for each product - **P**

USER AND PRODUCT COMBINED FEATURES

Reorders (User and Product) - (N)

- We determine how many times a User bought a specific Product - **N**

	user_id	product_id	up_purchase_count
0	1	196	10
1	1	10258	9
2	1	10326	1
3	1	12427	10
4	1	13032	3

- Calculate Max. Number of reorders by User (max of order number)

	user_id	uid_max_order_count
0	1	10
1	2	14
2	3	12
3	4	5
4	5	4

- With the above details, we check when the product was bought by the user for the first time (which order)

	user_id	product_id	up_order_count
0	1	196	1
1	1	10258	2
2	1	10326	5
3	1	12427	1
4	1	13032	2

- Merging above dataframes we get User X Product - first time order (number) and max purchases by user.

	user_id	product_id	up_order_count	uid_max_order_count
0	1	196	1	10
1	1	10258	2	10
2	1	10326	5	10
3	1	12427	1	10
4	1	13032	2	10

- Adding another column to above merged dataframe to show ‘no. of orders since first order’ of User X Product

	user_id	product_id	up_order_count	uid_max_order_count	ord_count_since_first_order
0	1	196	1	10	10
1	1	10258	2	10	9
2	1	10326	5	10	6

Keeping this dataframe on the left and merging it with Above dataframe, we arrive at all features pertaining to User X product.

	user_id	product_id	up_purchase_count	up_order_count	uid_max_order_count	ord_count_since_first_order
0	1	196	10	1	10	10
1	1	10258	9	2	10	9
2	1	10326	1	5	10	6
3	1	12427	10	1	10	10
4	1	13032	3	2	10	9

Then we calculate 'up_reorder_ratio' column which is **up_purchase_count/up_order_count_since_first_order**

```
user_prod_ratio['up_reorder_ratio'] = user_prod_ratio['up_purchase_count']/user_prod_ratio['ord_co
unt_since_first_order']
```

er_id	product_id	up_purchase_count	up_order_count	uid_max_order_count	ord_count_since_first_order	up_reorder_ratio
	196	10	1	10	10	1.000000
	10258	9	2	10	9	1.000000
	10326	1	5	10	6	0.166667
	12427	10	1	10	10	1.000000
	13032	3	2	10	9	0.333333

Extracting *user, Product, up_reorder_ratio* column from above dataframe to create new dataframe. Merging this dataframe with the Count of User X Product Purchase - we arrive at User X Product level total purchase count and reorder ratio for each product by a user.

	user_id	product_id	up_purchase_count	up_reorder_ratio
0	1	196	10	1.000000
1	1	10258	9	1.000000
2	1	10326	1	0.166667
3	1	12427	10	1.000000
4	1	13032	3	0.333333

For better prediction, the dataset is filtered to see if a product was reodered in the 5 most recent orders. Creating some statistics out of this value intuitively appears to help our model in determining which product is most likely to be bought in the next order.

To compute the above, we keep the last five orders for each user and get how many times a product was bought in those last 5 purchases. Doing this will give us a rich dataset of reordering users (of atleast 5 reorders) and the number of times a product was purchased in the last 5 orders.

We use the .transform() method to fill in the order numbers in reverse (i.e., incase a user has made 10 orders,we will consider the last(latest) 5 orders and the product ids related to those orders. So the order numbering will be reversed to filter out the most 5 recent orders. Using .transform() method will help us retain the size of the dataset and the aggregated values are updated in the assigned column.

```
order_merged['order_number_reversed'] = order_merged.groupby('user_id')['order_number'].transform
(max)\
                                                                                - order_merge
d['order_number'] +1
order_merged.head(5)
```

artment	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order	order_number_reversed
y eggs	202279	prior	3	5	9	8.0	6
duce	202279	prior	3	5	9	8.0	6
try	202279	prior	3	5	9	8.0	6
try	202279	prior	3	5	9	8.0	6
try	202279	prior	3	5	9	8.0	6

Using the above dataframe, we filter the 5 most recent orders (`order_number_reversed' <= 5`)

artment	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_orde	order_number_reversed
erages	22352	prior	4	1	12	30.0	5
sehold	22352	prior	4	1	12	30.0	5
sehold	22352	prior	4	1	12	30.0	5
erages	142903	prior	11	2	14	30.0	2
en	142903	prior	11	2	14	30.0	2

- We create a User x product dataframe by using user_id, product_id,up_order_count_5 (number of times product was ordered in the last 5 orders),
- Merging **N** (left join) with above dataframe to give user_id, product_id with purchase count and no. of times the item was reordered in last 5 orders.

In above arrived dataframe, we have NaN values in 'Last 5 orders' (because there were no reorders for that User X Product in last 5 orders). This is imputed with 0.

Merging **N** with **W** (left join) we arrive at dataframe - **S**

Merging above dataframe **S** with **P** (left join), we arrive at dataframe - **D**

By doing the above, we have User X Product based metrics

1. Count of each product purchase for each user (up_purchase_count)
2. Reorder probability (Ratio of number of times user bought a product to the total orders placed since first order of product)(up_reorder_ratio)
3. Number of times a product was reordered in the last 5 orders (up_order_count_5)
4. Maximum number of orders for each user (Max_no_of_orders)

5. How frequently a user reordered the products (user_reorder_ratio)
6. Number of reorders for each Product (product_purchase_count)
7. Probability of product to be reordered (prod_reord_probability)

	user_id	product_id	up_purchase_count	up_order_count_5	Max_no_of_orders	user_reorder_ratio
0	1	196	10	5.0	10	0.694915
1	1	10258	9	5.0	10	0.694915
2	1	10326	1	0.0	10	0.694915
3	1	12427	10	5.0	10	0.694915
4	1	13032	3	2.0	10	0.694915

product_purchase_count	prod_reord_probability
35791	0.776480
1946	0.713772
5526	0.652009
6476	0.740735
3751	0.657158

Creating feature set for our model

- For creating the feature set, we carry out the following steps:
- For creating our features dataframe, we create ‘future_orders’ dataframe by removing ‘prior’ & by selecting columns ‘user_id’, ‘eval_set’, ‘order_id’ - **F**
- We merge **D** with **F** on ‘user_id’, left join --also **D**
- We split train & test dataset from **D** --------we call it -**T**
- We merge **T** with ‘df_order_products_train’ (original train dataframe) taking ‘product_id’, ‘order_id’, ‘reordered’ columns joining (left join with **T** on left) on ‘product_id’ and ‘order_id’

In the above arrived dataframe, few changes are required to be carried out before feeding it into the model.

- Imputing NaN values to 0 in ‘reordered’ column
- Dropping ‘eval_set’, ‘order_id’ columns (unnecessary) from above dataframe.
- Setting ‘user_id’, ‘product_id’ as index for above dataframe.

With the above, out train dataframe is ready for modeling.

user_id	product_id	up_purchase_count	up_order_count_5	Max_no_of_orders	user_reorder_ratio	product_purchase_count	prod_reord_probability	reordered
1	196	10	5.0	10	0.694915	35791	0.776480	1.0
	10258	9	5.0	10	0.694915	1946	0.713772	1.0
	10326	1	0.0	10	0.694915	5526	0.652009	0.0
	12427	10	5.0	10	0.694915	6476	0.740735	0.0
	13032	3	2.0	10	0.694915	3751	0.657158	1.0

TEST DATAFRAME

- Create test dataframe -(**Te**)
- Dropping 'eval_set', 'order_id' columns (unnecessary) from above (**Te**) dataframe
- Setting 'user_id', 'product_id' as index for above dataframe.

user_id	product_id	up_purchase_count	up_order_count_5	Max_no_of_orders	user_reorder_ratio	product_purchase_count	prod_reord_probability	reordered
3	248	1	0.0	12	0.625	6371	0.400251	0
	1005	1	1.0	12	0.625	463	0.440605	0
	1819	3	0.0	12	0.625	2424	0.492162	1
	1819	3	0.0	12	0.625	2424	0.492162	1
	1819	3	0.0	12	0.625	2424	0.492162	0

Machine Learning

In our problem, we try to predict the next product in a user's instacart order. For this, we could use a simple logistic regression to get the probability of a product being in the User's next order. However, we utilize the powerful XGBoost Classifier.

XGBoost (Extreme Gradient Boosting) is an advanced implementation of Gradient Boosting Algorithm. Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher. This technique is followed for a classification problem while a similar technique is used for regression

The XGBoost Advantage

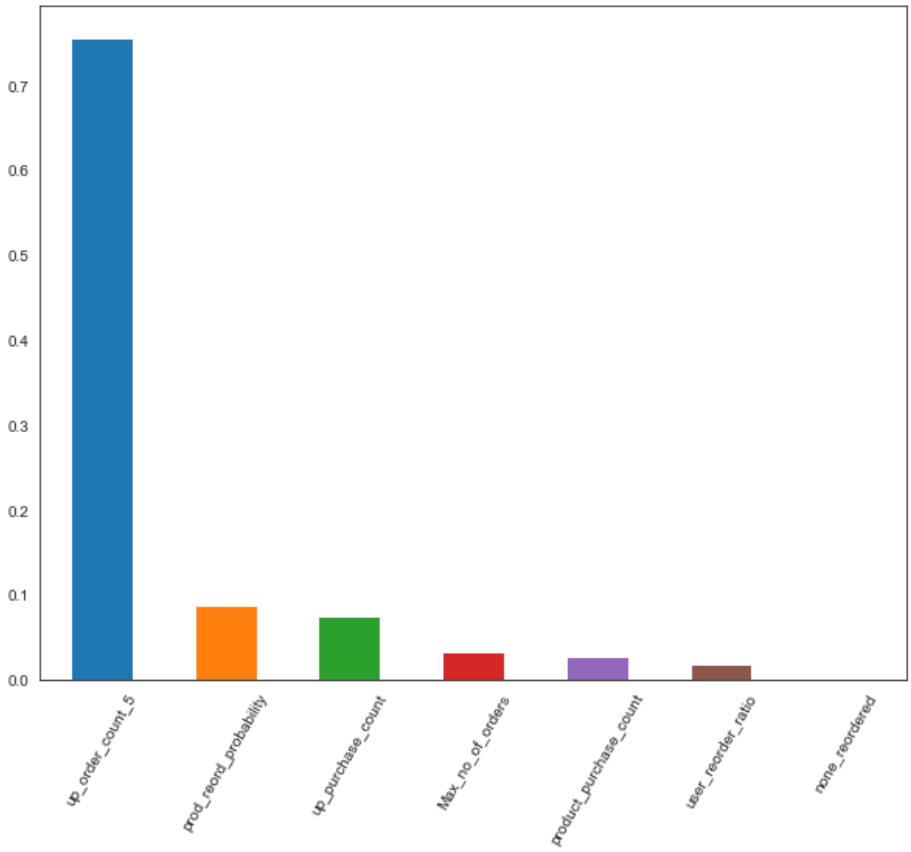
Using XGBoost has many advantages, as under :

1. Regularization:
    - Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.
    - In fact, XGBoost is also known as a ‘regularized boosting‘ technique.
2. Parallel Processing:
    - XGBoost implements parallel processing and is considered much faster as compared to GBM.
    - XGBoost also supports implementation on Hadoop.
3. Handling Missing Values
    - XGBoost has an in-built routine to handle missing values.
    - The user is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
4. Built-in Cross-Validation
    - XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
5. Continue on Existing Model
    - User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.

Modeling

Iteration 1

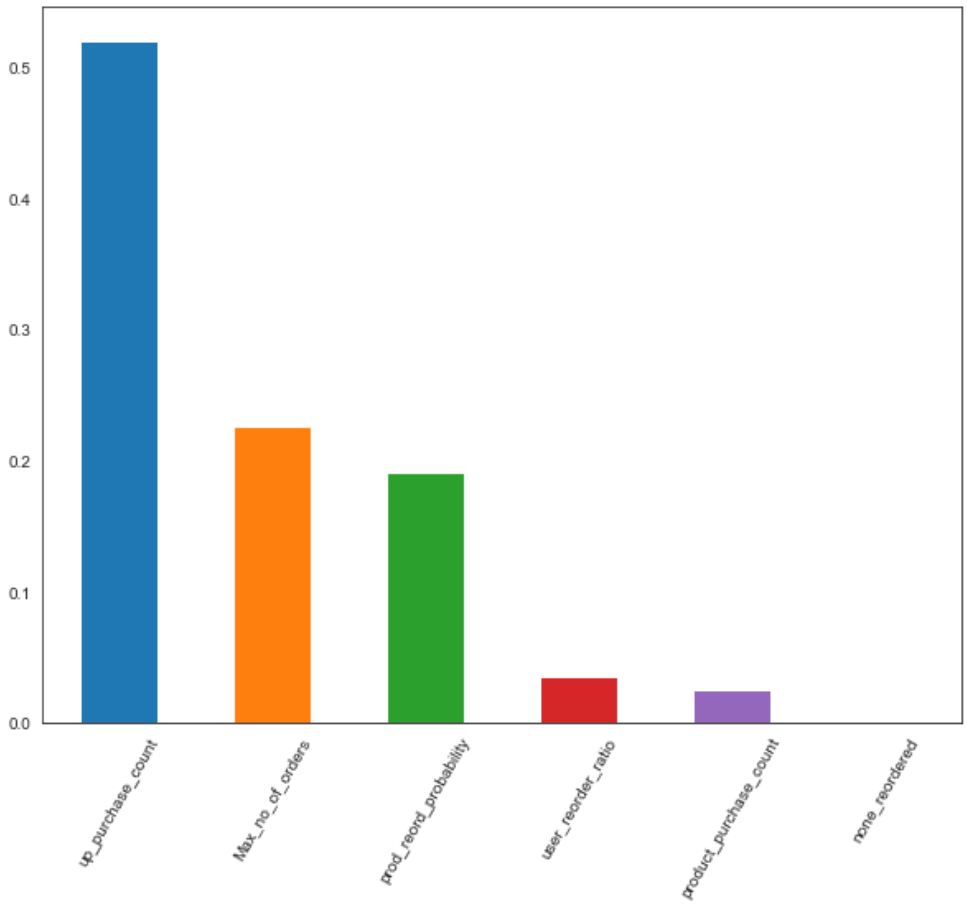
- Creating X_train, y_train from **T** dataframe
- Instantiating XGBClassifier, fit and check feature importances
- Predict_proba to get exact probabilities



As it is observed from the above graph, the up_order_count_5 is the most important feature among the above features arrived at , which try to undermine the other features' importances. Intuitively, a product repeatedly being purchased in the last 5 orders tend to be in the user's next order. By removing this feature, we try to arrive at the feature importance which is more generalized.

Iteration 2

We remove the most prominent feature from the 1st iteration which is the 'up_order_count_5' and fit the model on our data again, we arrive at the feature importances which better represent the reordering criteria.



Predicting NONE reorder probability

In the above model, we tried to find the ‘reorder’ probability. From the ‘reorder’ column, we try to create a new feature ‘NONE’ reorder by applying a threshold to ‘prod_reord_probability’ column (0 if x<= 0.5 else 1). With the above threshold values, we create new feature (column) ‘none_reordered’. Here 0 indicates NO reorder and 1 indicates reorder.

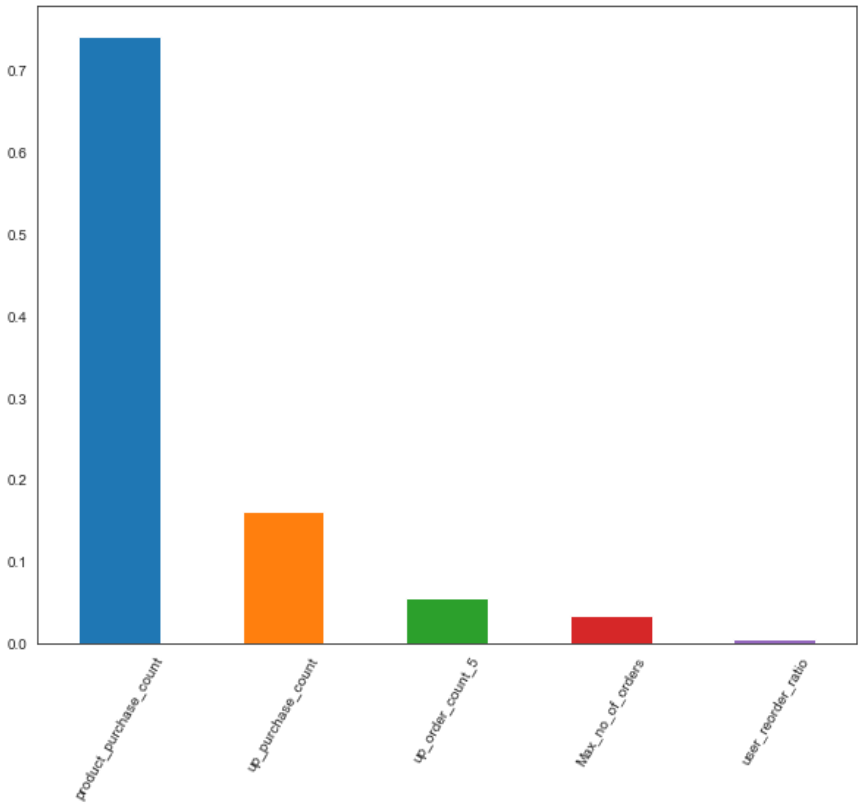
```
train_dataset['none_reordered'] = train_dataset['prod_reord_probability'].apply(lambda x: 0 if x <=0.5 else 1)
```

```
train_dataset.head(5)
```

user_id	product_id	up_purchase_count	up_order_count_5	Max_no_of_orders	user_reorder_ratio	product_purchase_count	prod_reord_probability	reordered
1	196	10	5.0	10	0.694915	35791	0.776480	1.0
	10258	9	5.0	10	0.694915	1946	0.713772	1.0
	10326	1	0.0	10	0.694915	5526	0.652009	0.0
	12427	10	5.0	10	0.694915	6476	0.740735	0.0
	13032	3	2.0	10	0.694915	3751	0.657158	1.0

Applying the same threshold to test dataset, we create the dependent variable ‘none_reordered’ to the Test and train datasets.

We run the above model using the XGBClassifier. The feature importances, as expected, differ from the importances arrived at by using the ‘reorder’ model (in iteration 1 and 2).



Running mode with ‘NONE’ training set and using .predict_proba, we arrive at the probabilities and a better F1 score than the initial ‘reorder’ model. (improvement from 0.68 to 0.87)

```
y_pred2 = (xgb2.predict_proba(X_test_none)[:, 1] >= 0.22).astype('int')


print('F1 Score for Reorder prediction: {}'.format(f1_score(y_pred2, y_test_none)))
print(classification_report(y_pred2, y_test_none))
```

```
F1 Score for Reorder prediction: 0.8736382203229602
              precision    recall  f1-score   support

           0       0.09      0.86      0.17    296051
           1       1.00      0.78      0.87  11496447

    accuracy                           0.78  11792498
   macro avg       0.54      0.82      0.52  11792498
weighted avg       0.97      0.78      0.86  11792498
```

FINAL MODEL

In our final model, we combine the probabilities arrived at the first model (Iteration 2) of 'reorder' and the probabilities arrived at 'none_reorder' prediction and multiplying them.

```
pred_final = np.multiply(y_pred_prb_1, y_pred_prb_2)
```

```
pred_final
```

```
array([[0.35112038, 0.02858863],
       [0.64264387, 0.01610352],
       [0.29534176, 0.11737714],
       ...,
       [0.40061605, 0.01458715],
       [0.73641205, 0.00219556],
       [0.4174007 , 0.01307027]], dtype=float32)
```

For the final model, we give minimal parameters while running a GridSearch Cross validation (minimal parameters to have lesser computation time) to arrive at an F1 score of 0.75 with accuracy of 0.74

```
F1 Score for Combined model prediction: 0.7522835339991514
              precision    recall  f1-score   support

           0       0.82      0.64      0.72   6188611
           1       0.68      0.84      0.75   5603887

    accuracy                           0.74  11792498
   macro avg       0.75      0.74      0.73  11792498
weighted avg       0.75      0.74      0.73  11792498
```

LIMITATIONS

Limitations of the above model :

1. Only limited features were extracted using the user x product combination. Inclusion of several other features pertaining to date/day/time of purchase/Department/Aisle etc could probably enhance the dataset and give out better accuracy in the final model but it was not tested due to time and computational constraints.
2. XGBoost Classifier was used which is considered robust and fast among other Classification algorithms. We took F1 score as metric as it takes into consideration precision as well as recall. Other ensemble models were not tested which may or may not perform better than the ones used here.
3. GridSearch Cross Validation and parameter tuning for the ‘reorder’ model and no fine tuning of parameters for ‘NONE’ reorder model : These were not carried out due to time and computational constraints, carrying out the same might affect the F1 score values of the final outcome.
4. While calculating y_pred values for Iteration 2 of our first model, we have set a threshold of 0.22, tweaking that value might affect the final outcome of our model, which was not tested in this project.
5. Only two models were considered in this project. In an ideal scenario, multiple models with different feature sets and combining their values using weighted averages will result in a better final outcome (F1 score of final model) but was not carried out due to time and computational constraints.