Problem

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. After selecting products through the Instacart app, personal shoppers review your order and do the in- store shopping and delivery for you. Instacart uses customers' transactional data to develop models which recommend products that users will buy again based on their previous purchases.

The goal of the project is to predict which previously purchased products will be in a user's next order. By providing an optimal and accurate recommendations of products to purchase, Instacart can enhance customers overall shopping experience, browsing experience, increase revenue from sales, and increase overall customer satisfaction.

Data

The data can be obtained from kaggle, https://www.kaggle.com/c/instacart-market-basket-analysis/data. The dataset is a relational set of files describing Instacart customers' orders over time. The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, Instacart provides between 4 and 100 of their orders, with the sequence of products purchased in each order. Instacart also provide the week and hour of day the order was placed, and a relative measure of time between orders.

The data comprises of six different csv files and most of them are self-explanatory. Snapshot of the csv files are included below.

aisles.csv:
aisle_id,aisle
 1,prepared soups salads
 2,specialty cheeses
 3,energy granola bars
 ...

departments.csv:
department_id,department
 1,frozen

2,other
3,bakery
…

order_products__*.csv:
These files specify which products were purchased in each order.
order_products__prior.csv contains previous order contents for all customers.
'reordered' indicates that the customer has a previous order that contains the product.
Note that some orders will have no reordered items.
order_id,product_id,add_to_cart_order,reordered
1,49302,1,1
1,11109,2,1
1,10246,3,0
...

orders.csv:
This file tells to which set (prior, train, test) an order belongs. 'order_dow' is the day of week.
order_id,user_id,eval_set,order_number,order_dow,order_hour_of_day,days_since_prior_order
2539329,1,prior,1,2,08,
2398795,1,prior,2,3,07,15.0
473747,1,prior,3,3,12,21.0
...

products.csv:
product_id,product_name,aisle_id,department_id
1,Chocolate Sandwich Cookies,61,19
2,All-Seasons Salt,104,13
3,Robust Golden Unsweetened Oolong Tea,94,7
...

Since the dataset was obtained from Kaggle, the data is already pretty clean. The only missing values are inside the days_since_prior_order column of orders.csv. There are some “NaN” values in that column which most likely represent the first order of a particular user. Since it’s the first order, there is no prior order so days_since_prior_order is “NaN”.

```
print(orders.info())
orders.head()
```

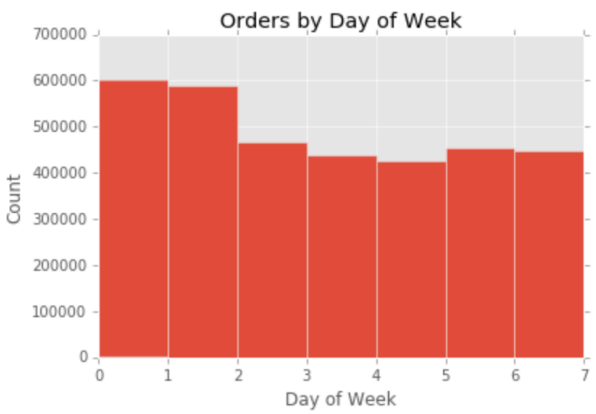
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3421083 entries, 0 to 3421082
Data columns (total 7 columns):
order_id                int64
user_id                 int64
eval_set                object
order_number            int64
order_dow               int64
order_hour_of_day       int64
days_since_prior_order  float64
dtypes: float64(1), int64(5), object(1)
memory usage: 182.7+ MB
None
```

	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

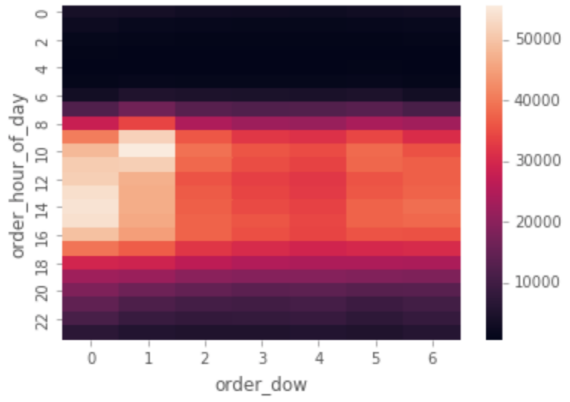
The “NaN” comprises about 6% of total number of rows in the orders data. When doing the EDA involving the “days_since_prior_order” column, these “NaN” data are excluded. In the modeling part of the project, these “NaN” can be replaced by “-1” and the model should be able to learn what is the meaning of “-1”.

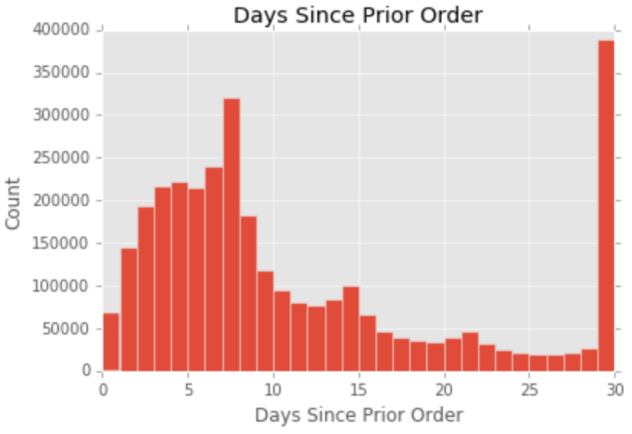
Data Exploration

Analysis of Orders Data:
From the plot below, we can see that Day 0 and Day 1 have the most number of orders. It is unclear what days Day 0 and Day 1 represent. The peak hours are between 9AM and 5PM but from this plot we cannot really see clearly the corresponding day and time combination.



Use heatmap (below) instead. From the heatmap, the peak day and hours combination occurs on day 0 and day 1 between 9AM and 5PM. From the Days Since Prior Order plot, we can see that a lot of customers put another order after a week or a month which makes sense because people tend to reorder after a week or a month.





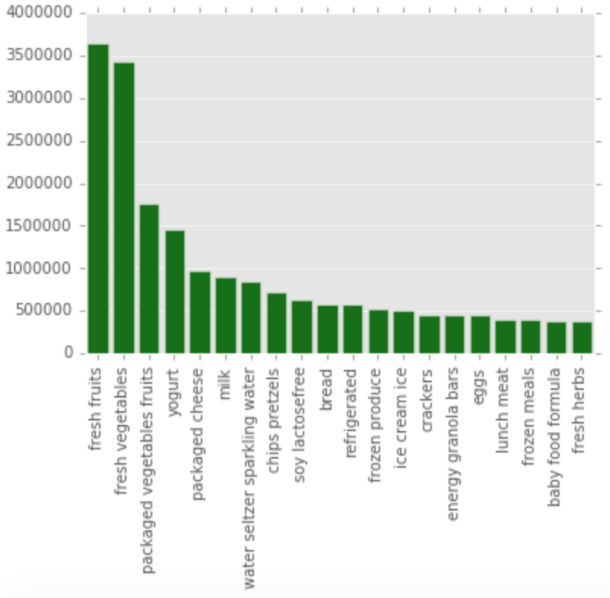
Analysis of Products Data:

Product	Count
Banana	472565
Bag of Organic Bananas	379450
Organic Strawberries	264683
Organic Baby Spinach	241921
Organic Hass Avocado	213584
Organic Avocado	176815
Large Lemon	152657
Strawberries	142951
Limes	140627
Organic Whole Milk	137905
Organic Raspberries	137057
Organic Yellow Onion	113426
Organic Garlic	109778
Organic Zucchini	104823
Organic Blueberries	100060
Cucumber Kirby	97315
Organic Fuji Apple	89632
Organic Lemon	87746
Apple Honeycrisp Organic	85020
Organic Grape Tomatoes	84255

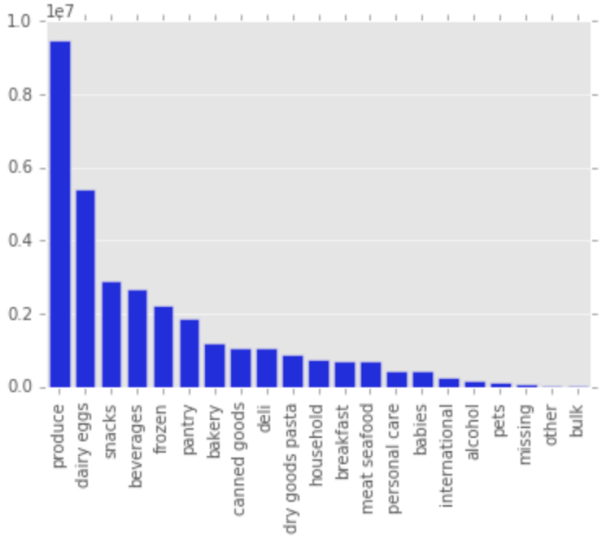
Name: product_name, dtype: int64

From the table above, the top 20 products are mostly fruits and vegetables (except Whole Milk). The bar plot of top aisles below confirms that the top products and aisles are those of fruits and vegetables. From the bar plot of top departments below, produce department dominates orders which is consistent with fruits and vegetables as the top products.

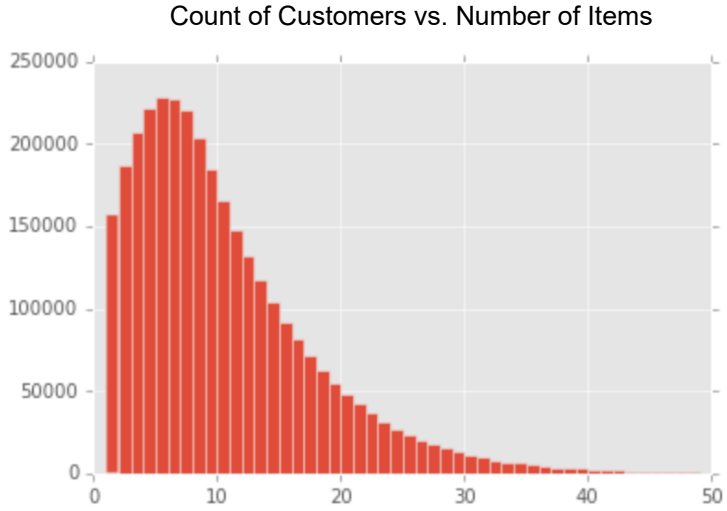
Top 20 Aisles



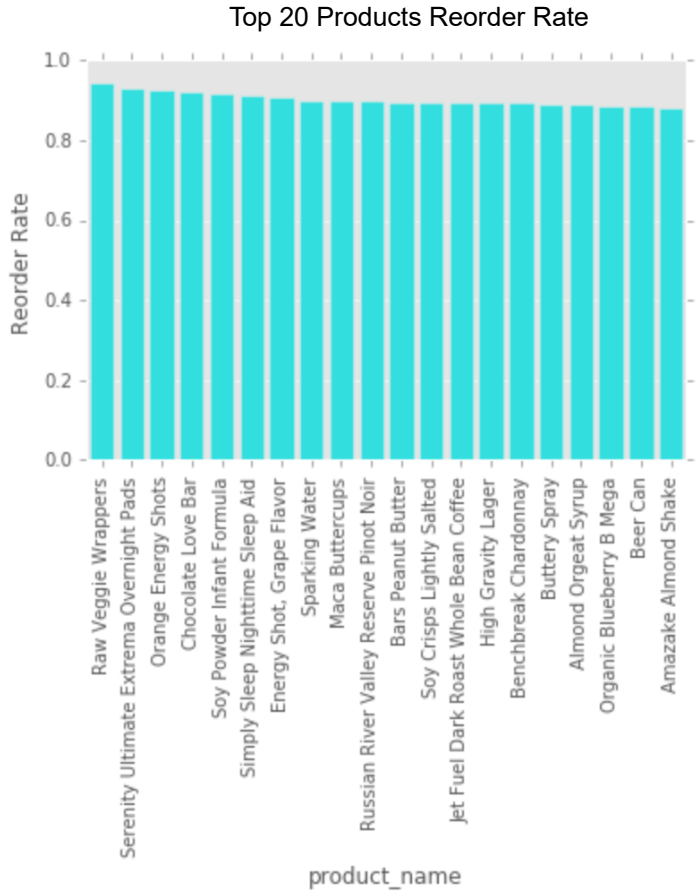
Top 20 Departments



The bar plot below shows the count of customers (y-axis) that buy how many items (x-axis). A lot of customers bought 4 to 7 products per order.



The bar plot below shows top 20 products that has the highest reorder rate (~90%).



Features Extraction

To produce the final data frame that we can feed to the machine learning model, we need to extract features from different datasets (orders, products, users, aisles, departments). The features are divided into different level of aggregation. There are user level features, order level features, product level features, and user-product level features in the final data frame.

The user level features are (mapped by userid, unique per user):

- user_total_orders: Total number of orders for a specific user.
- user_total_items: Total number of items bought by a specific user.
- total_distinct_items: Total number of distinct items bought by a specific user.
- user_average_days_between_order: Average days between order of a specific user.
- user_average_basket: Average basket/cart size of a specific user.

The order level features are (mapped by orderid, unique per order):

- order_hour_of_day: Order hour of the day.
- days_since_prior_order: Number of days since prior order.
- days_since_ratio: days_since_prior_order/user_average_days_between_order.

The product level features are (mapped by productid, unique per product) :

- aisle_id: Aisle id of a specific product.
- department_id: Department id of a specific product.
- product_orders: Total number of orders for a specific product.
- product_reorders: Total number of reorders for a specific product.
- product_reorder_rate: product_reorders/product_orders.

The user-product level features are (mapped by userid-productid combination, unique per user-product combination):

- userproduct_orders: Total number of orders for a specific user-product combination.
- userproduct_orders_ratio: userproduct_orders/user_total_orders
- userproduct_average_pos_in_cart: Average position in cart of a product for a specific user.
- userproduct_reorder_rate: Reorder rate for a specific user-product combination.
- userproduct_orders_since_last: The difference between the number of total orders for a specific user and the order number of the last order id for a specific

user-product combination. In other words, this is the distance between the last order of a user and a user last order a particular product.
- userproduct_delta_hour_vs_last: Time difference between a specific order hour of the day and the last time a particular user order a particular product.

Train - Test Split

Since kaggle does not provide the testing data, the original training data is split into new training data (80%) and new testing data (20%) so that we can fit the model using the new training data measure the performance of the machine learning model on the new testing data.

Labels

The final data frame (training or testing) is labelled 1 or 0 (binary) based on whether or not the combination of (order_id, product_id) from all possible products is inside the data frame. Labels variable is the actual target variable (we will compare this to the predicted variables).

For example, for the training data, using the code below:
labels += [(order_id, product_id) in train.index for product in user_products]
The index of train dataset is the order_id and product_id. Here, the code is checking whether or not that combination of (order_id, product_id) is inside the training data.

Light GBM

The model used in this problem is Light GBM. To understand Light GBM, we need to understand gradient boosting. There are two main components involved in gradient boosting: a loss function to be optimized (auc, log-loss, etc.) and a weak learner (decision tree). Gradient boosting is an additive model where trees are added one at a time and a gradient descent procedure is used to minimize the loss when adding trees. The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve final output of the model.

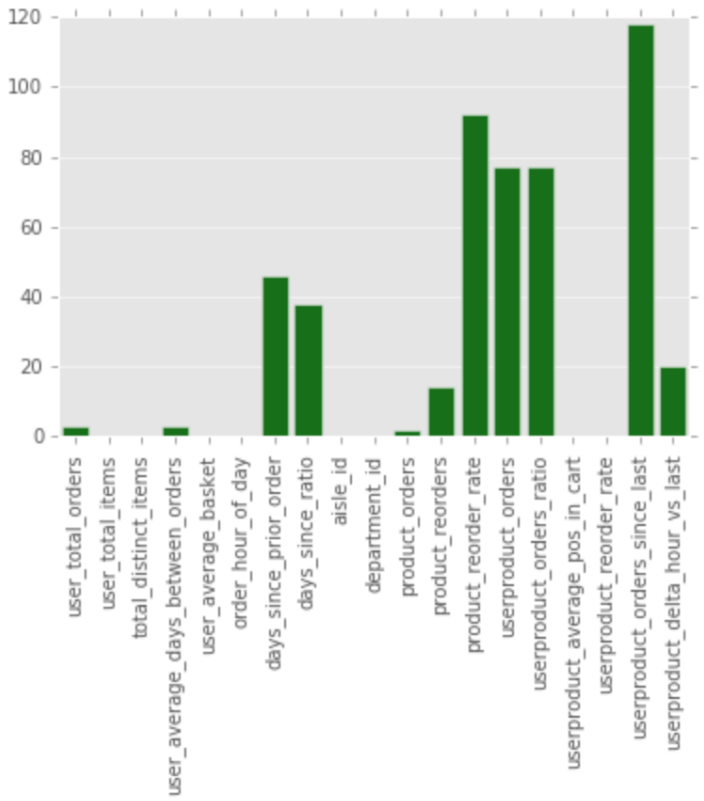
Light GBM is relatively new and is a type of gradient boosting like XGBoost but with much faster run time and comparable performance. Instead of splitting the tree depth wise (XGBoost), Light GBM splits the tree leaf wise which results in a faster execution time. Splitting the tree leaf wise may lead to increase in complexity (and overfitting) but this can be overcome by tuning the parameter max-depth.

The main parameters of Light GBM are:

- Number of leaves: Main parameter that controls the complexity of the model.
- Max depth: Used to limit the tree depth and prevent overfitting
- Learning rate: A high learning rate can lead to overfitting but a lower learning rate is slower.

Model Implementation

The training data frame (all features, labels) is fitted to a baseline (using standard parameters) Light GBM model to obtain the feature importances plot below.



From the feature importances plot above, we can see that a few features are not important and the important features are:

- days_since_prior_order

- Days_since_ratio
- product_reorders
- product_reorder_rate
- userproduct_orders
- userproduct_orders_ratio
- userproduct_orders_since_last
- userproduct_delta_hour_vs_last

Then, a new Light GBM model is trained using these important features only. While training the data, a 5 folds Grid Search CV is also performed to determine which parameters combination produces the best model (the highest accuracy). The parameters tuned are: learning rate, number of leaves, and max depth. The best parameters combination is:

- learning rate = 0.1
- number of leaves = 80
- max depth = 8

Using the best parameters combination, we refit the model and use the new model to predict the probability of a specific user-product combination. The probability is then converted to a 1 or 0 (whether or not a specific user-product is in the data frame) based on a certain probability threshold.

The optimum probability threshold of 0.32 is obtained by trying different probability values and computing the auc, accuracy, and f1 score for each possible probability value. The optimum threshold is chosen to be the one that has the highest f1 score and relatively high auc and accuracy.

The final performance of the model on the test data is:

- AUC = 0.697
- Accuracy = 0.876
- F1 Score = 0.430