Instacart Market Basket Analysis

——Data Mining Final Project Report

Team 1

Prepared by

Ya Liu
Qing Ruan
Zixuan Huang

November 26, 2019

Table of Contents

1. Introduction

Instacart released a dataset, “The Instacart Online Grocery Shopping Dataset 2017”, which contains a sample of over one million grocery orders from more than 200,000 Instacart users. For each user, they provided their orders and the sequence of products in each order. They also provided the time including the week and the hour of day the order was placed, and the relative time between orders. Using this data to test models, Instacart will be able to predict products that a user will buy again. Our project using this dataset from Kaggle aims to predict which previously purchased products would be in a consumer’s next order by constructing various classification models.

The structure of this report is described as follow. The 2nd section shows an overview of the dataset and exploration of data. In section 3, we describe data mining algorithms we used in this project. Section 4 shows whole experiment process from feature selection to different models building at last. These models include Logistic Regression, KNN and Random Forest. In section 5, we will discuss all results of our experiments in more detail. The last part in section 6 presents the summary for our results and a potential improvement to the project.

1. Description of Data

2.1 Source of Data

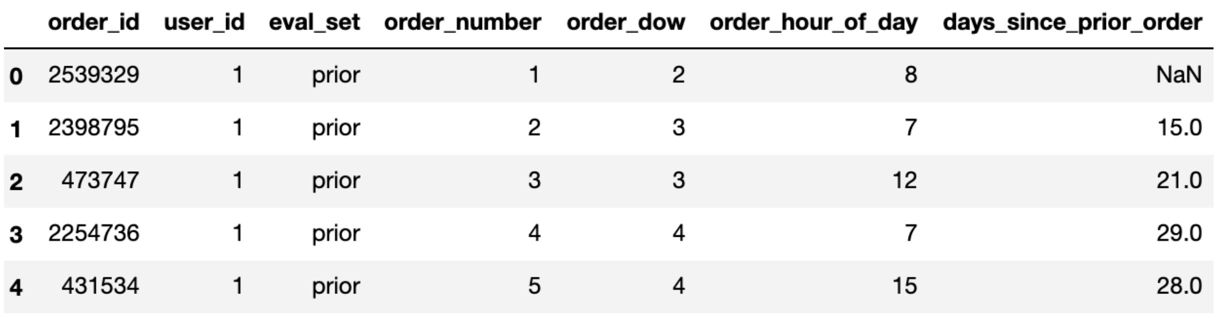
The source datasets for our analysis are relational sets of .csv files from Kaggle competition website. The dataset consists of information about 3.4 million grocery orders, distributed across 7 csv files.

2.2 Overview of Dataset

Now, we display all of the *.csv files forming the raw data.

2.2.1 Orders.csv

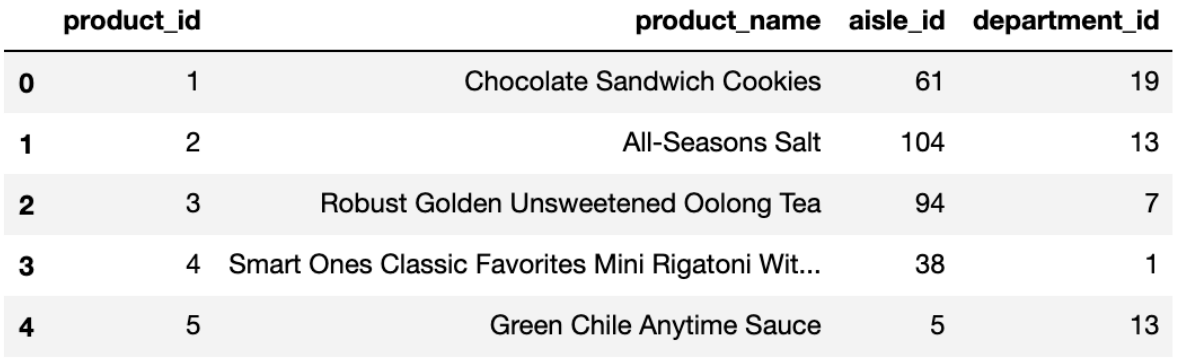
A list of all orders and 1 row for per order. It tells which set (prior, train and test) an order belongs. `order_dow` is the day of week.



	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
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

2.2.2 Products.csv

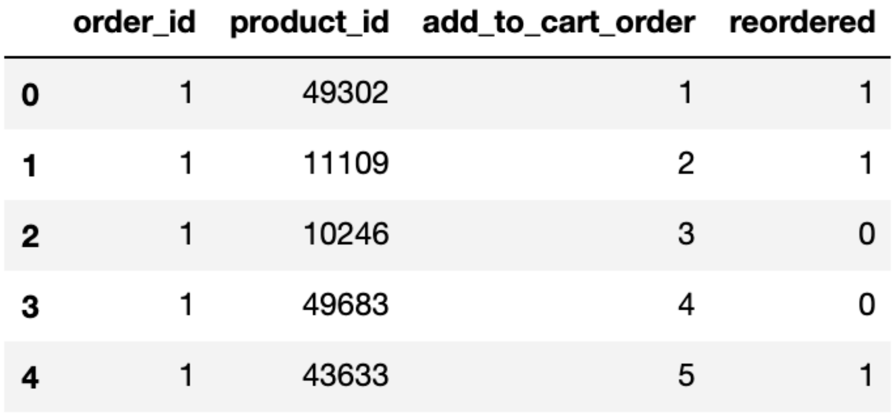
A file contains names of the products with their corresponding prouct_id. We can know about which products were ordered from `order_product_trian.csv`, and it also contains information of the order in column `add_to_cart_order` in which the products were put into the cart and information of whether this product is `re-ordered`.



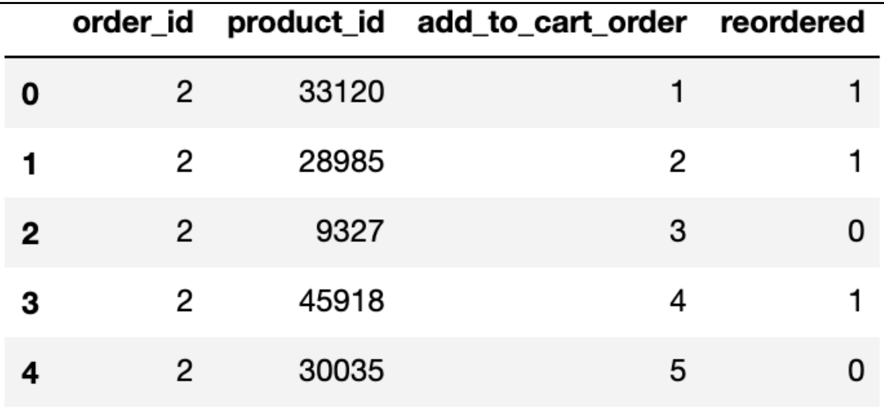
	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

2.2.3 Order_products_*(train/prior).csv

Structurally, these two files are the same except that the prior.csv records previous orders but the train.csv records the last order of each user.



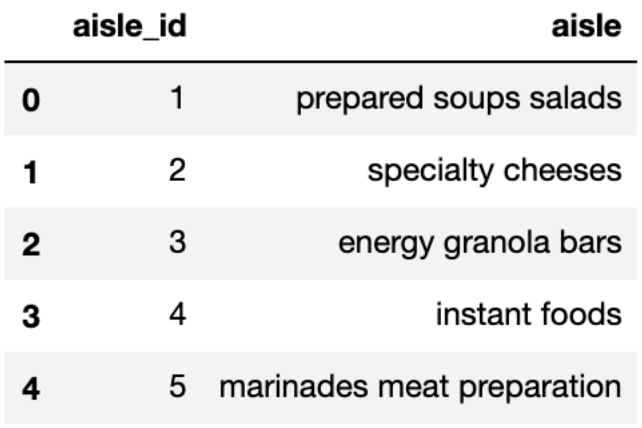
	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1



	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

2.2.4 Aisles.csv

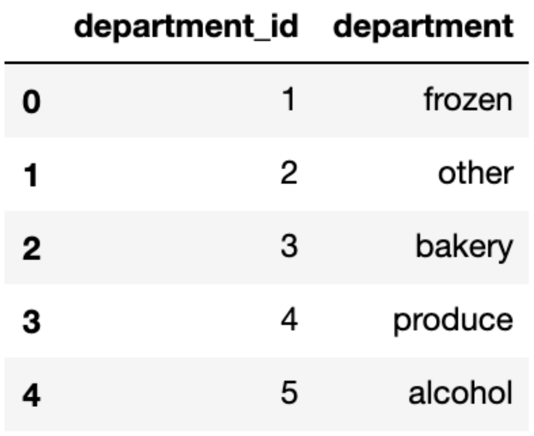
This file shows aisles information including aisle id and aisle name.



	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

2.2.5 Departments.csv

This file shows departments information including department id and department name.



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

In the datasets, we can find that each entity (customer, product, order, aisle, etc.) has an associated unique id.

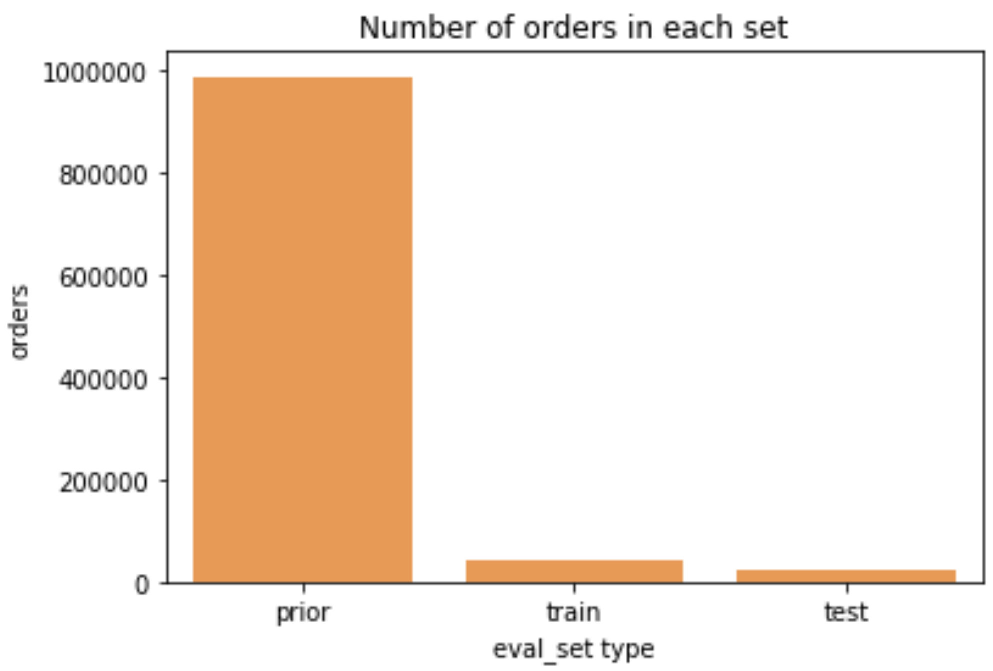
2.3 Exploration of Data

2.3.1 Orders Information

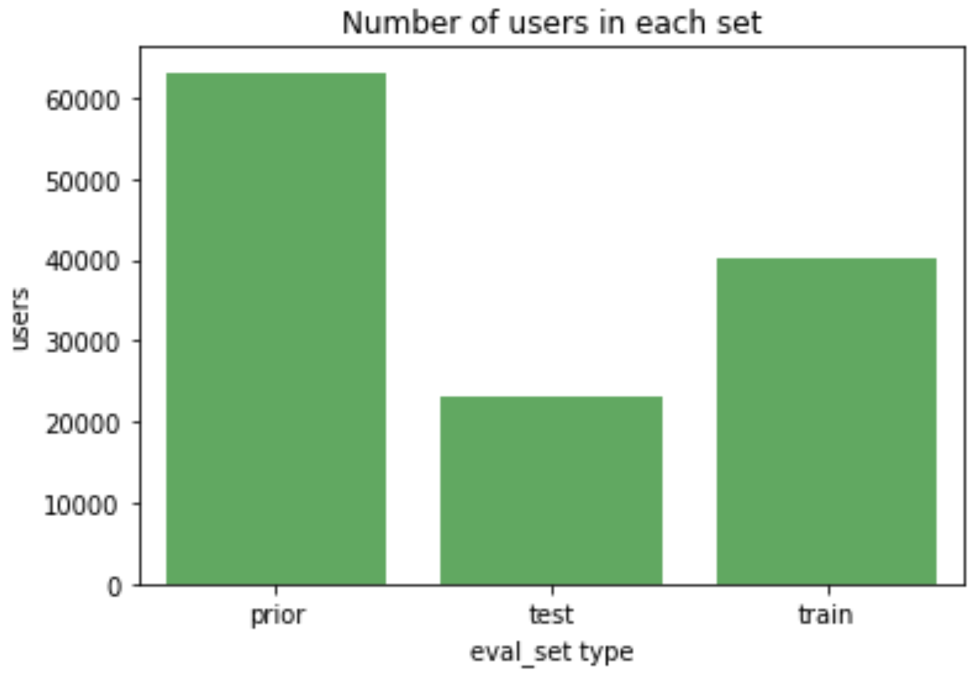
- Three Sets

Firstly, we do the basic exploration for this csv. There are 3,421,083 observations totally. We check the missing values and data types for each column. The `days_since_prior_order` has missing values for the reason that the first order didn't have this attribute. Thus, we replace these Nan with zero instead later.

Now, we take a look at some characteristics about three sets.

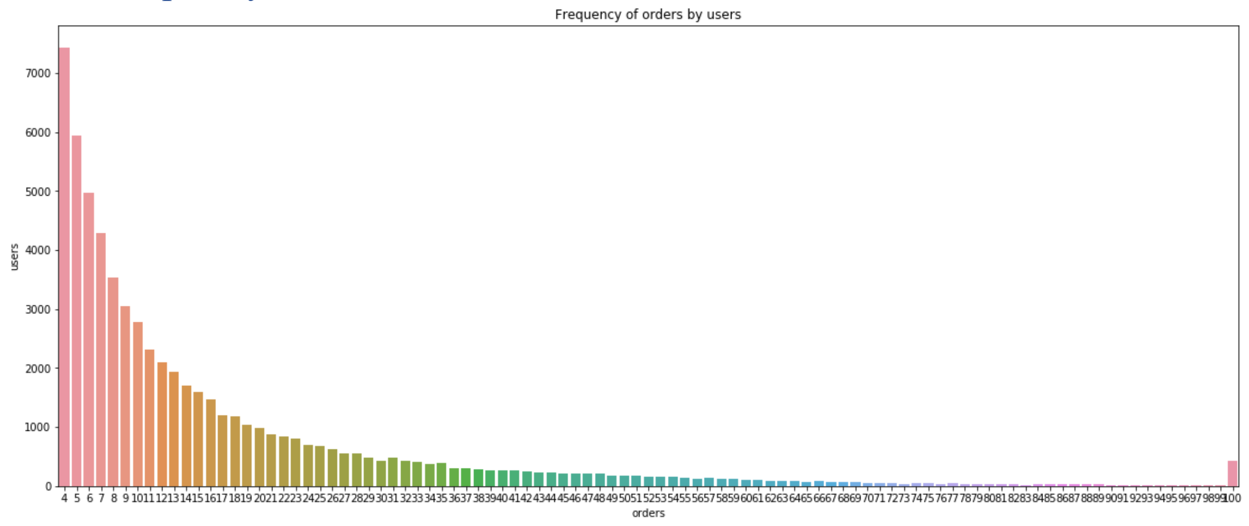


There are 985475 orders for the prior set, and the dataset extract the last order of each customer as train and test dataset, respectively. The train set has 40096 observations and the test dataset has 23004 observations.



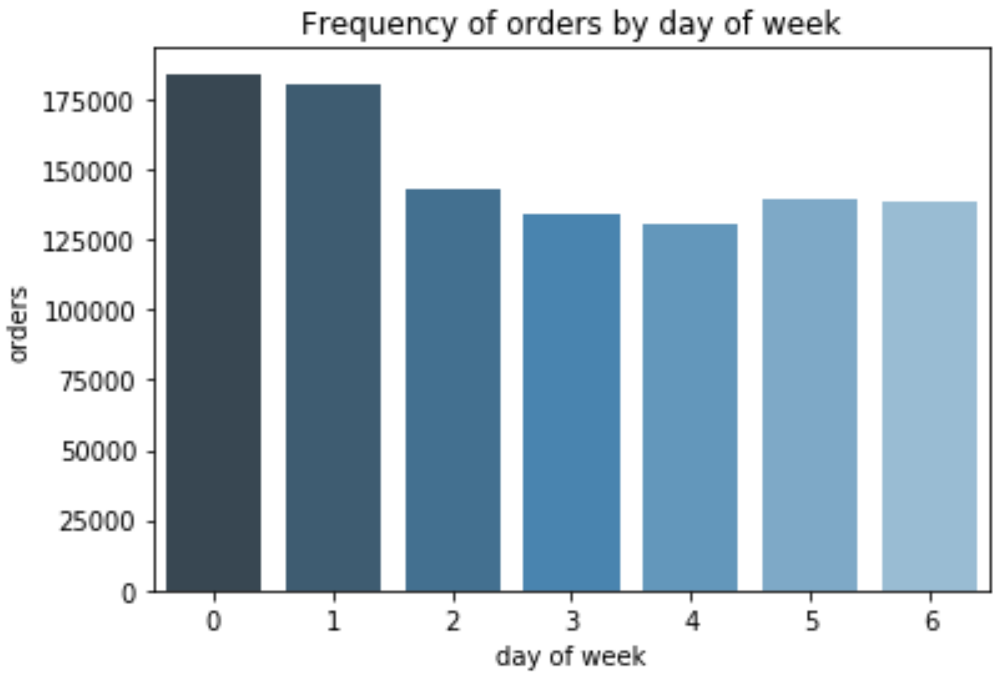
There are 63100 users in prior set, 23004 users in test set and 40096 users in train set.

- Frequency of Orders

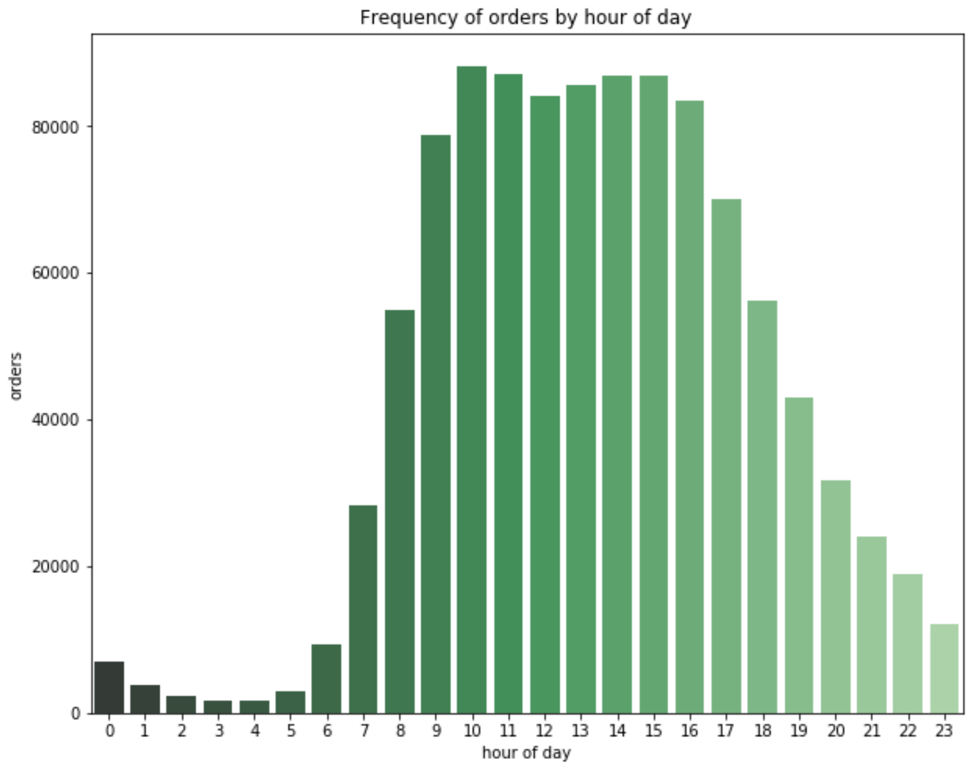


There are no orders less than 4 and is max capped at 100 as given in the data page.

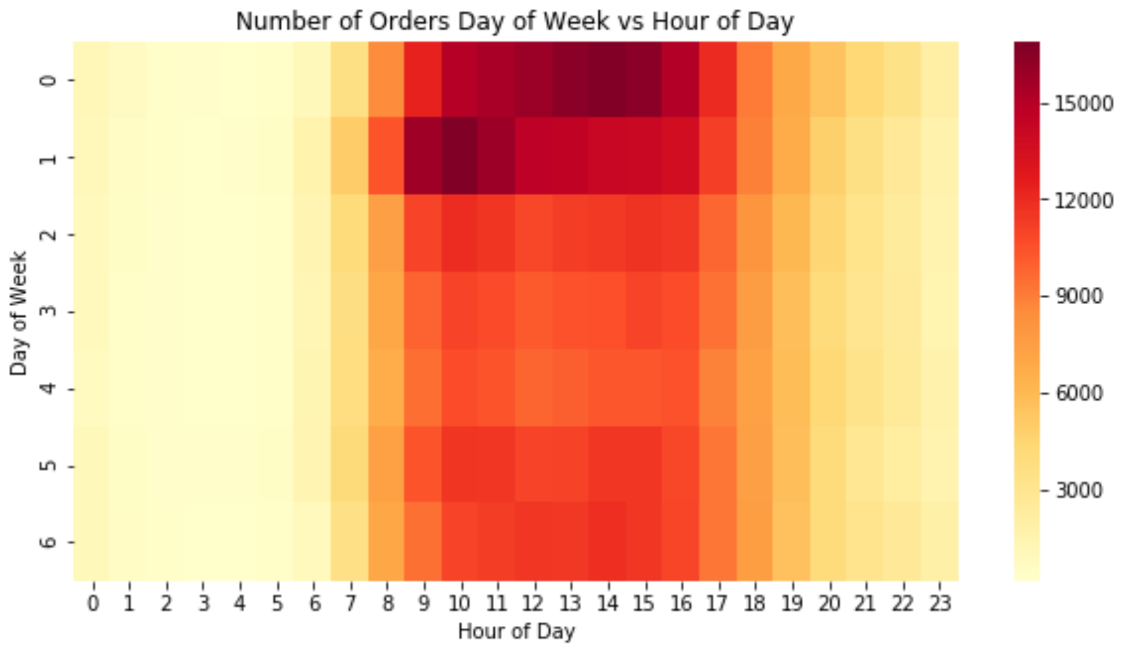
- Time of Orders



It looks as though 0 represents Saturday and 1 represents Sunday. Wednesday is then the least popular day to make orders.

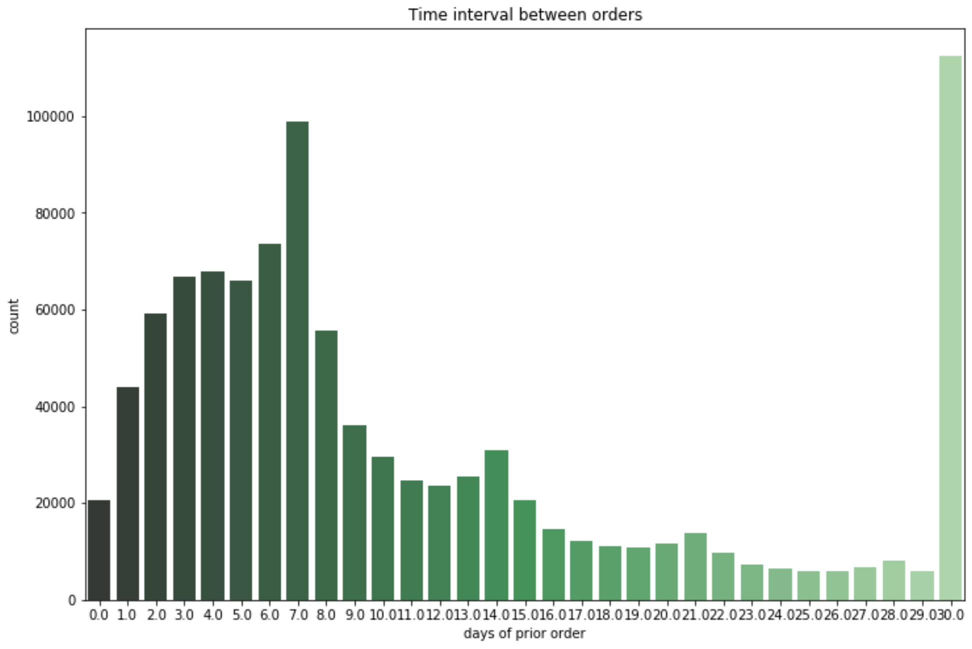


So, majority of the orders are made during daytime. The 10am hour is the most popular time to make orders, followed by a dip around lunch time and a pickup in the afternoon. Now let us combine the day of week and hour of day to see the distribution.



Saturday afternoon and Sunday morning are the most popular time to make orders.

- Time Interval between Orders

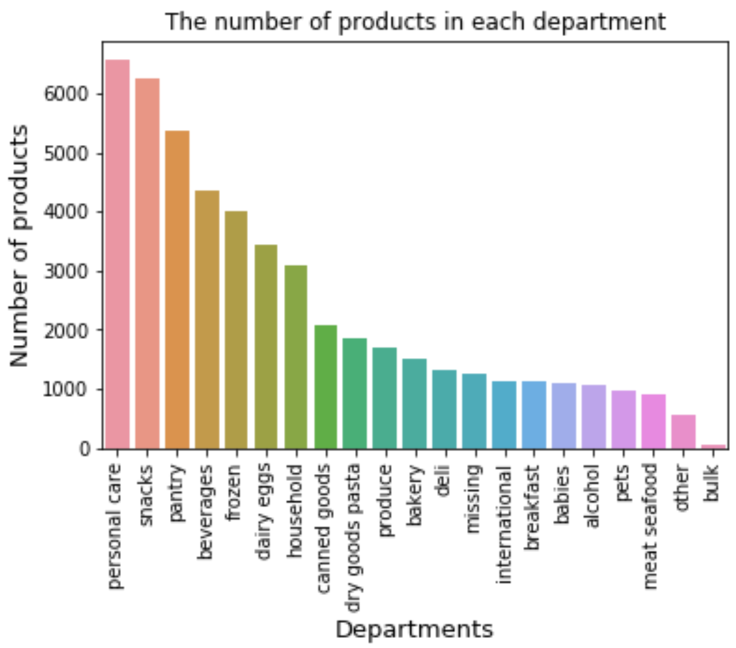


While the most popular relative time between orders is monthly (30 days), there are "local maxima" at weekly (7 days), biweekly (14 days), triweekly (21 days), and quadriweekly (28 days).Therefore it appears that customers order once in every week (check the peak at 7 days) or once in a month (peak at 30 days). We could also see smaller peaks at 14, 21 and 28 days (weekly intervals).

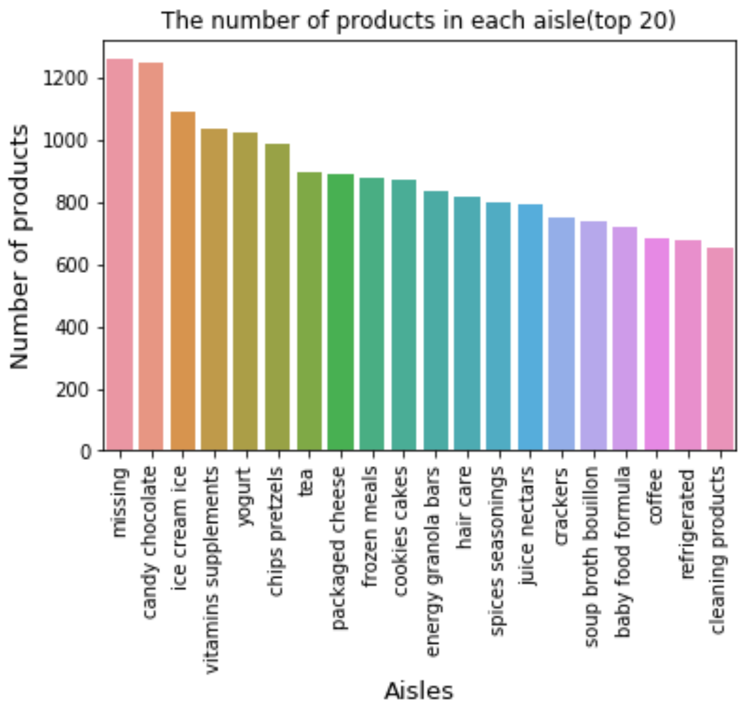
2.3.2 Product Information

- Product in Departments and Aisles

In this part, we will explore more on product information. Here we merge products.csv, aisles.csv and departents.csv.



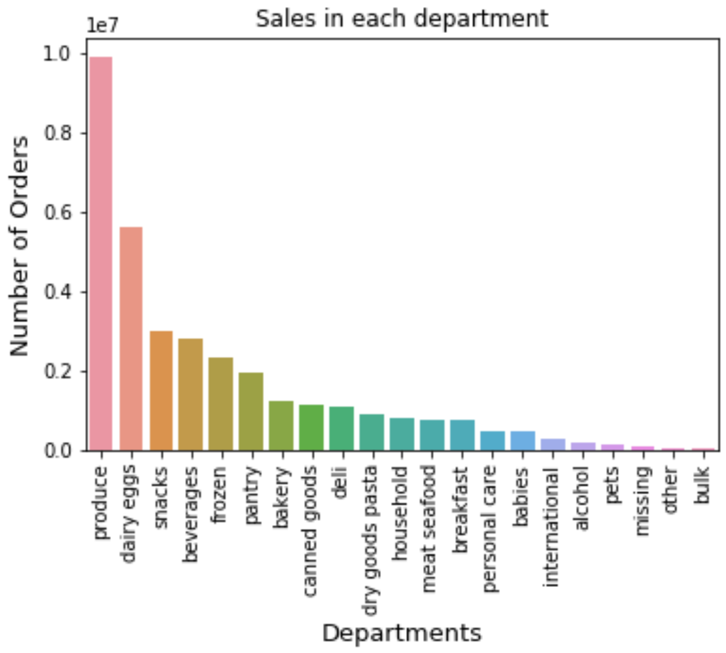
The most five important departments are personal care, snacks, pantry, beverages and frozen. The number of items from these departments were more than 4,000 times.



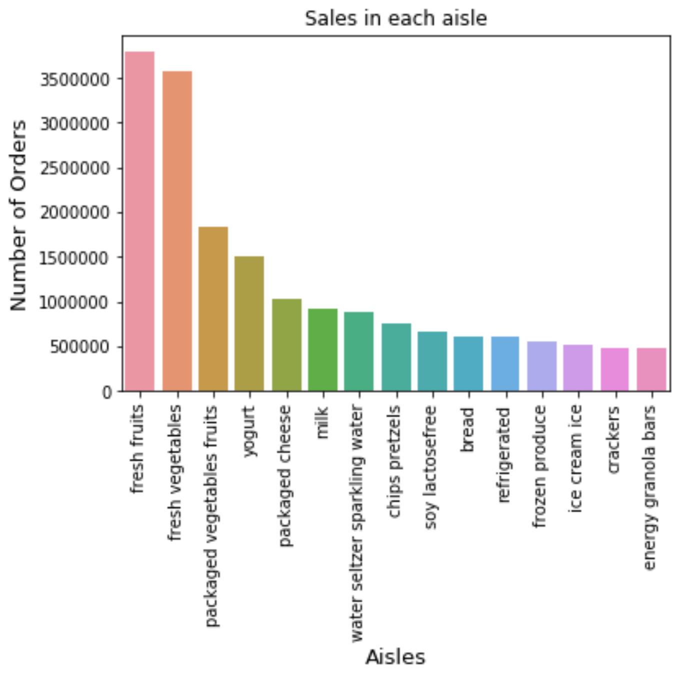
The most three important aisles are candy chocolate, ice cream and vitamins supplements.

- Sales in Departments and Aisles

Here we merge products.csv, order_products_*(train/prior).csv and orders.csv to get more information.



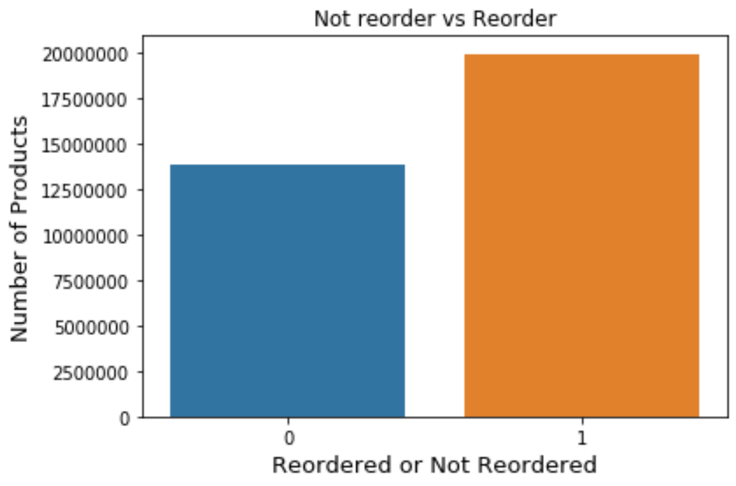
The most three popular departments are produce, dairy eggs and snacks.



The top three best-selling aisles are fresh fruits, fresh vegetables and packaged vegetables fruits.

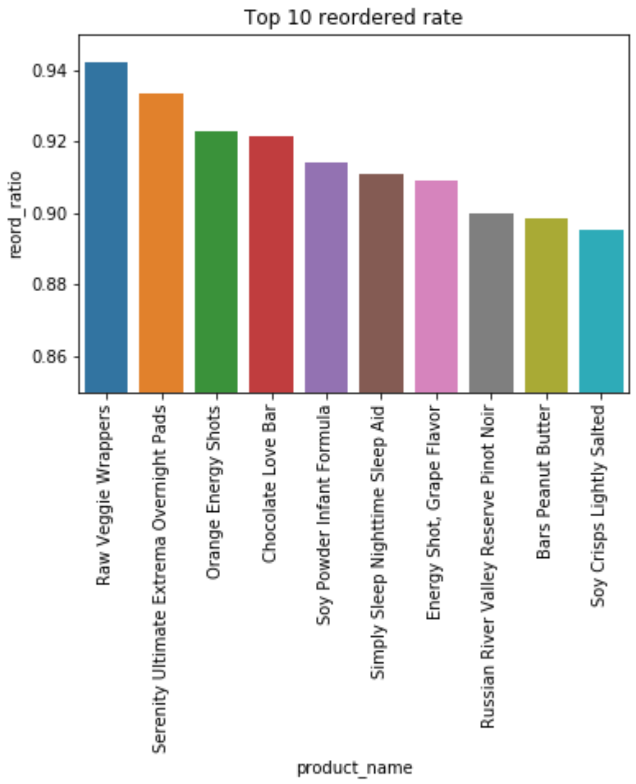
- Reorder of Products

First, we take a look at the number of reordered products.



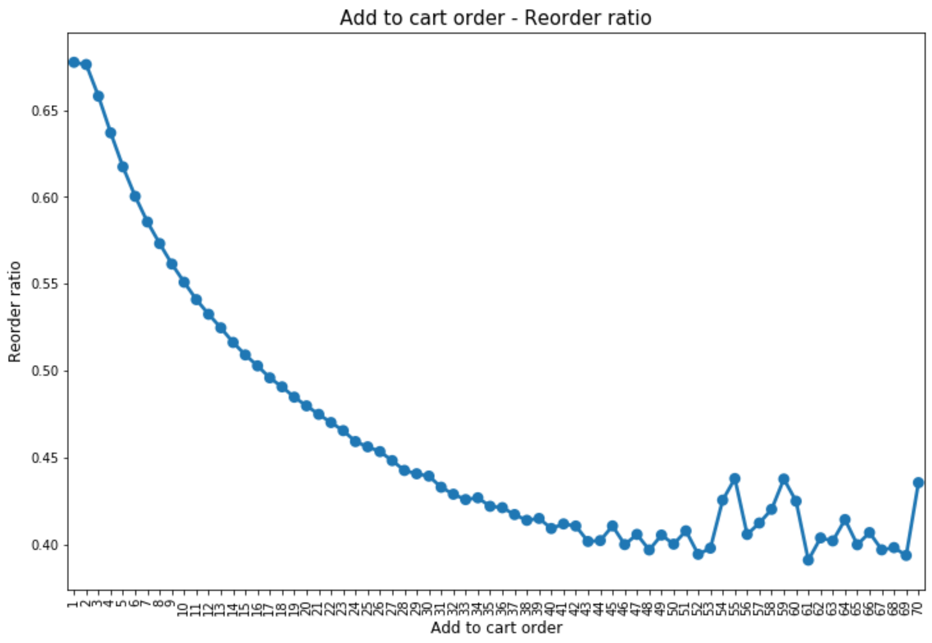
There are 19,126,536 products previously ordered by customers, reordered products take 0.59 % of ordered products and 13,307,953 products are not ordered by customers before, non-reordered products take 0.41 % of ordered products.

Next is the highest reordered rate.



The three products with the highest reordered rate are Raw Veggie Wrappers, Serenity Ultimate Extrema Overnight Pads and Orange Energy Shots.

Lastly, let us find some relationship between add_to_cart and reordered.



From the graph above, we can find that orders placed initially in the cart are more likely to be reordered than the one placed later in the cart.

Besides, we did t-test to verify whether the sequence of adding to cart are significantly different between reordered products and not reordered products. The results show that the p-value is smaller than 0.05. Therefore, the sequence of adding to cart significantly influence whether the products being reordered.

2. Data Preprocessing

Firstly, we performed a check on `order_id` for both train/prior set and orders set. And we find that the `order_id` for both train & prior dataset and orders set is correspondent one by one, which is consistent with data description in the kaggle website overview.

Luckily, in the datasets we use, all of them have no missing value except the orders.csv with `days_since_prior_order`. There are 206209 missing values in days_since_piror_orders, they mean these orders are first order for the user, respectively. According to this meaning, we replace Nan with -1 to indicate that it is a different level.

3. Features Engineering

We create a new Dataframe named `merge_oop` combining `orders` and `order_products_prior`. Here `merge_oop` only contains prior observations.

As for training, all of the data we used to analyze should come from prior relevant information excluding the last order information for each user, for the next feature selection part, we only use 'merge_oop' we created just now instead of orders and order_products_prior.

Similarly, 'order' dataset includes prior orders and last order used for training label. Therefore, we need to exclude information about last order for each user before using this dataset.

Now, we have two dataframes which are merge_oop and orders_prior.

Then we can create features using the merged dataframes we created before. There are three types of features we created as below. We will introduce them in more detail as below.

4.1 User Feature (`user`)

It describes the patterns of one user. There are seven sub features in this type. Here we use data from `merge_oop` dataframe we set before to create them.

(1) `n_orders_users`: The number of orders for each user.

Created a new dataframe named by `user_f_1`. Acquired by grouping users (`user_id`) from `merge_oop`and calculate the number of orders (`order_number`).

(2) `n_products_users`: The number of products for each user.

Create a new dataframe named by `user_f_2`. Acquired by grouping users (`user_id`) from `merge_oop` and calculate the number of products (`product_id`).

(3) `avg_products_users`: Average products of one order for each user.

Created a new column in `user_f_2`. Acquired by dividing `user_f_1` by `n_products_users` created above.

(4) `dow_most_user`: The day in one week ordered most for each user.

Created a new dataframe named by `user_f_3`. Calculate the number of orders for each day of week and select the day corresponding to the maximum count.

Numeric variable.

(5) `times_h`: The time of one day ordered most for each user.

Create a new dataframe named by `user_f_4`. Calculate the number of orders for different hour in one day from `merge_oop` dataframe and select the hour corresponding to the maximum count.

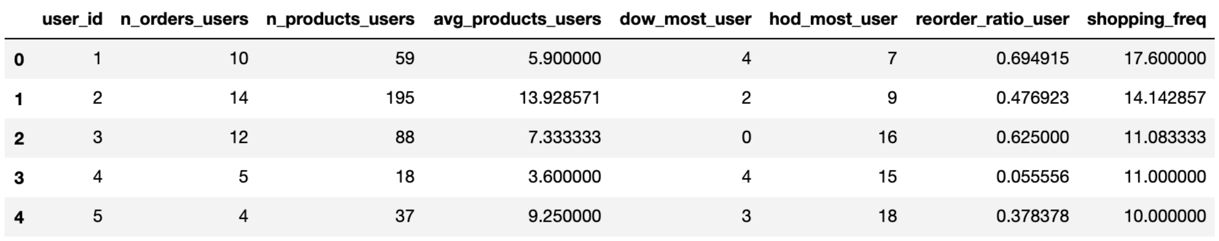
(6) `reorder_ratio_user`: Reorder ratio for each user.

Create a new dataframe named by `user_f_5`. Group `merge_oop` dataframe by 'user_id'. As the values for `reordered` columns is 0 or 1, we create reordered ratio by calculating the mean of it for each group.

(7) `shopping_freq`: Shopping frequency for each user.

Create a new dataframe named by `user_f_6`. Group `merge_oop` dataframe by 'user_id' and calculate the mean of intervals between orders (`days_since_prior_order`) for each user.

Now, joining on common `user_id` column, we merge all `user_f_*` (* from 1 to 6) into one dataframe and name it with `user` just like below.



	user_id	n_orders_users	n_products_users	avg_products_users	dow_most_user	hod_most_user	reorder_ratio_user	shopping_freq
0	1	10	59	5.900000	4	7	0.694915	17.600000
1	2	14	195	13.928571	2	9	0.476923	14.142857
2	3	12	88	7.333333	0	16	0.625000	11.083333
3	4	5	18	3.600000	4	15	0.055556	11.000000
4	5	4	37	9.250000	3	18	0.378378	10.000000

4.2 Product Feature

It describes the characteristics of one product. There are four sub features in this type. Here we use data from `order_products_prior` dataframe we set before to create them.

(1) `times_bought_prod`: Times of ordered for each product.

Create a new dataframe named with `prod_f_1`. Acquired by grouping product (``product_id) and calculate the count of orders (`order_id`).

(2) `reorder_ratio_prod`: Reordered ratio for each product.

Create a new dataframe named with `prod_f_2`. Acquired by grouping product (``product_id) and calculate the mean of reordered.

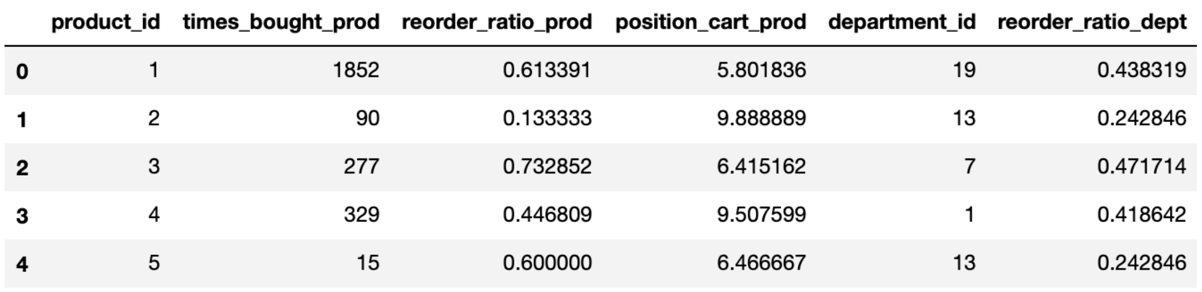
(3) `position_cart_prod`: Average positions of cart for each product.

Create a new dataframe named with `prod_f_3`. Acquired by grouping product (``product_id) and calculate the mean of cart-position (`add_to_order`) for each group.

(4) `reorder_ratio_prod`: Reorder ratio for each department.

Create a new dataframe named with `prod_f_4`. Merge `department`, `product` and `prod_f_3` with `product_id`, group this dataframe by departments and calculate the mean of `reorder_ratio_prod` in each group.

Now, joining on common `product_id` column, we merge all `prod_f_*` (* from 1 to 4) into one dataframe and name it with `prod` just like below.



	product_id	times_bought_prod	reorder_ratio_prod	position_cart_prod	department_id	reorder_ratio_dept
0	1	1852	0.613391	5.801836	19	0.438319
1	2	90	0.133333	9.888889	13	0.242846
2	3	277	0.732852	6.415162	7	0.471714
3	4	329	0.446809	9.507599	1	0.418642
4	5	15	0.600000	6.466667	13	0.242846

4.3 User & Product Feature

It describes the pattern of one user for specific product. There are three sub features in this type. Here we use data from `merge_oop` dataframe we set before to create them.

(1)`times_bought_up`: Times of one product bought by one user.

Create a new dataframe named with `user_prd_f_1`. Acquired by grouping user (`user_id`) and product (`product_id`) and calculate the count of orders (`order_id`).

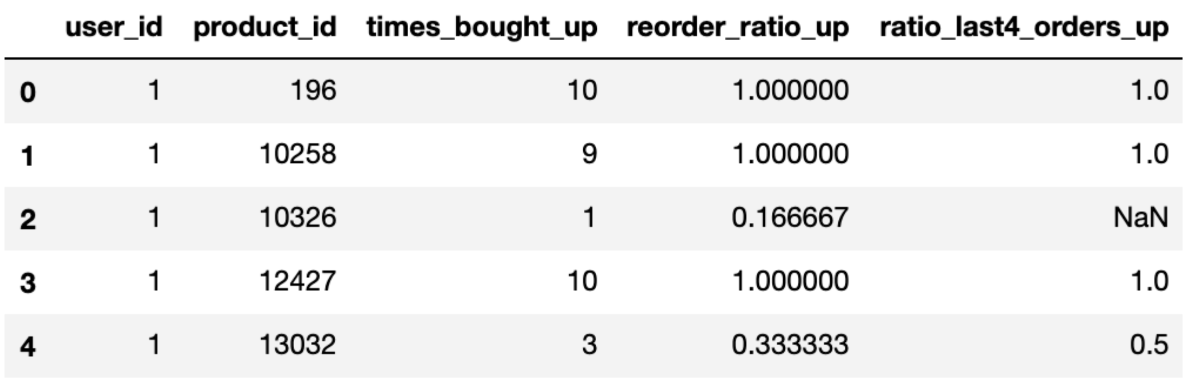
(2) `reorder_ratio_up`: Reorder ratio of one product bought by one user.

Create a new dataframe named with `user_prd_f_2`. Before calculating, group `merge_oop` dataframe by user and product. First for one product create the number of orders since the user bought it for the first time (`order_range`) by subtracting order number first bought (minimum of the `order_number`) with number of orders. Then create the times of bought for this product (`times_bought_up`) and acquire the reorder_ratio_up by dividing `order_range`.

(3) `ratio_last4_orders_up`: Ratio of one product bought in one user's last four orders.

Create a new dataframe named with `user_prd_f_3`. Reverse the order number first for each product and then create a new column `order_number_back` which selects orders where order_number_back is smaller than 5. Group `merge_oop` by user and product and acquire this feature by calculating the count of `order_number_back` dividing four. This new dataframe is named with `user_prod_f_3` including `user_id`, `product_id` and `ratio_last4_orders_up`.

Now, joining on common `user_id` and `product_id` column, we merge all `user_prod_f_*` (* from 1 to 3) into one dataframe and name it with `user_prod` just like below.



	user_id	product_id	times_bought_up	reorder_ratio_up	ratio_last4_orders_up
0	1	196	10	1.000000	1.0
1	1	10258	9	1.000000	1.0
2	1	10326	1	0.166667	NaN
3	1	12427	10	1.000000	1.0
4	1	13032	3	0.333333	0.5

After checking, we notice that some rows for raio_last4_orders_up have NaN values for our new feature. This happens as there might be products that the customer did not buy on its last four orders. For these cases, we turn NaN values into 0.

4.4 Merge All Features

Total features can be got by merging the three types of features in dataframes above and we name the new dataframe including all of features with `total_info`.

Finally, we select 14 features to do our models further. They are `n_orders_users`,`n_products_users`,`avg_products_users`,`dow_most_user`,`hod_most_user`,`reorder_ratio_user`,`shopping_freq`,`product_id`,`times_bought_up`,`reorder_ratio_up`,`times_last4_orders_up`,`times_bought_prod`,`reorder_ratio_prod`, `position_cart_prod`, `department_id`, `reorder_ratio_dept`.

4. Get Features and Target

We create the train/test dataset on the basis of `total_info` dataframe by adding dependent variable `reordered`.

First, by selecting eval_set equals to train, we acquire the last `order_id` information for each user. Then, we add them to the `total_info` dataframe. On the other side, we only select `product_id`,`order_id` and `reordered` columns from `order_products_train` dataframe.

Notice that, here `reordered` means if in this user's last order, this product has been bought before. Merge it with `total_info` and select merging on `total_info` by common columns `order_id` and `product_id`. After merging, only the product occurred in prior orders can be merged, therefore only reordered ==1 can be got, and for other products the user did before but not for the last order, the reorder is Nan. And we replace Nan with zero, which means this product did not occur in this user's last order.

As each user_id and product_id is unique which means one specific product for one certain user, we set these two elements as index.

After preprocessing, we get the final dataset used to build our models named with `total_info_train`. It has 2,588,813 observations and 15 columns with 14 features and one dependent variable.

5. Oversampling

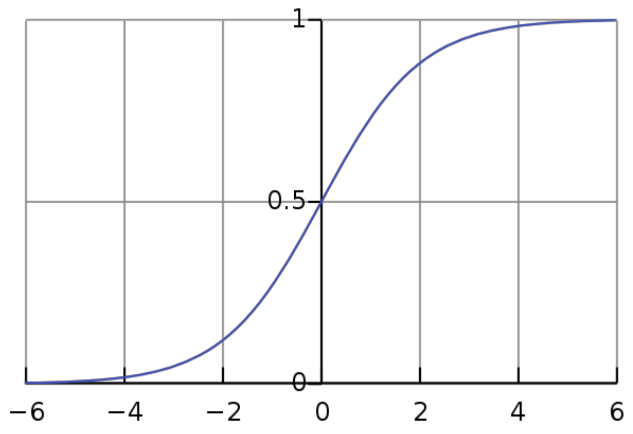
We observe that our target has two levels, 0 and 1, with frequency of 2,334,883 and 253,930, respectively. Level 0 account for 90% of all, which means this is a biased dataset. Therefore, we use oversampling to correct this by increasing the frequency of level 1 to the be the same as that of level 0. In the end, level 0 and level 1 have the same frequency 2,334,883. It shows that oversampling would significantly boost our model.

6. Baseline Model: Logistic Regression

7.1 Principle

In this project, our objective is to predict which product will be reordered in one user next order and it belongs to a classification problem. Therefore, we will use data mining algorithms focusing on solving problems with two class values.

Logistic regression is named for the function used at the core of the method, the logistic function. This function curve is shown as below. It is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.



Here is the logistic function equation:

$$f(x)=1/(1+e^{(-x)})$$

Logistic regression uses this equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights to predict an output value (y). Here the output value being modeled is a binary value (0 or 1). For example:

$$P(x)=e\hat{}(b_0+b_1\times x)\ )/(1+e\hat{}(b_0+b_1\times x)\ )$$

If $P(x) < 0.5$, binary value is 0, else is 1.

Here key for this algorithm is the coefficients (beta values), which can be acquired from our training data. Next, we will use this logistic regression to build our model and make predictions.

7.2 Develop Logistic Model

First, we remove `reordered` column from total_info_train dataset as X dataset, and `reordered` alone as Y. Split train and test using sklearn package from python. Here the test size, we set 0.2, which means 80% of our observations are used as train and 20% of them are used as test. They are named with X_train and X_test.

Then we scale X_train and X_test by creating class StandardScaler and using fit function.

After train and test split and scaling X data, we build logistic regression model. All of the parameters we select default mode.

7.3 Model Evaluation

7.3.1 Model Evaluation Statistics

(1) Accuracy

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

$$Accuracy = TP+TN / TP+TN+FP+FN$$

Here TP means true positives, TN means true negatives, FP means false positives and FN means false negatives.

(2) Precision

Precision means the ratio of correctly predicted positive observations (TP) to the total predicted positive observations (TP+FP).

$$Precision = TP / TP+FP$$

(3) Recall

Recall means the ratio of correctly predicted positive observations to the all observations in actual class.

$$Recall = TP / TP + FN$$

(4) F-1 score

It means the harmonic mean of precision and recall taking both metrics into account in the following equation:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

(5) ROC_AUC Score

The AUC is the area under the ROC curve. It shows the tradeoff between sensitivity and specificity.

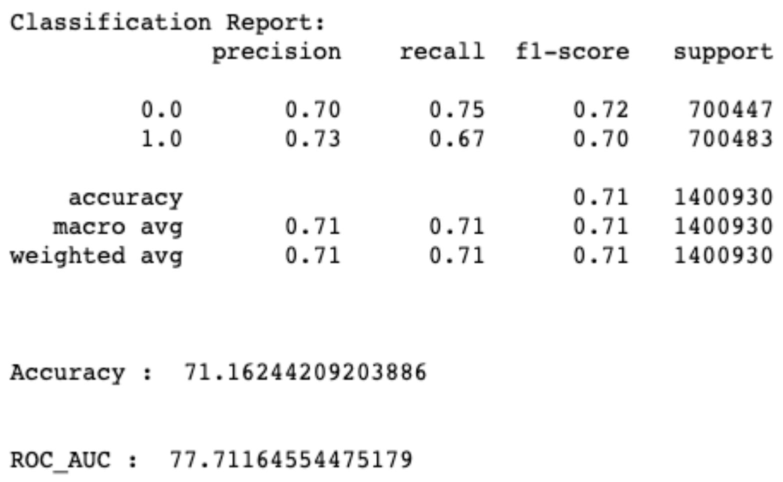
And roc_auc score gives us a good idea of how well the model performances. The AUC value lies between 0.5 to 1 where 0.5 denotes a bad classifier and 1 denotes an excellent classifier.

(6) Confusion Matrix

From the confusion matrix, we can get a better idea of what our model is getting right and what types of errors it made.

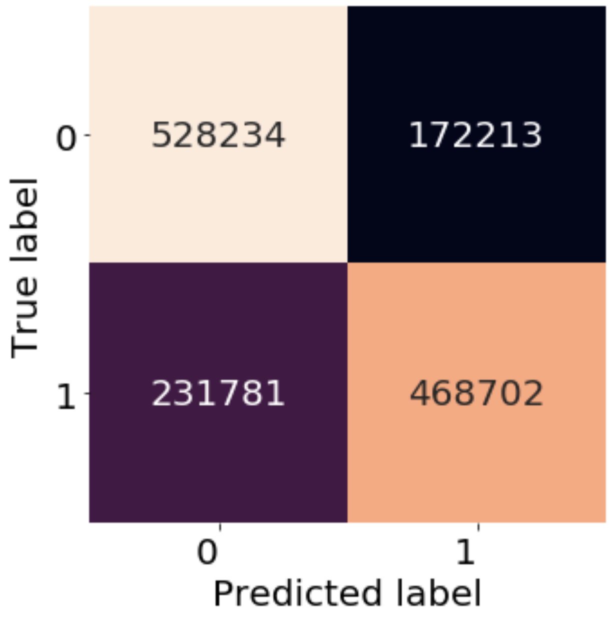
7.3.2 Evaluation of Logistic Model

From the classification report below we can see that the accuracy is 71.18%, which means the model can correctly classify 71.18% of the test dataset.



```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.70      0.75      0.72    700447
         1.0       0.73      0.67      0.70    700483

    accuracy                           0.71   1400930
   macro avg       0.71      0.71      0.71   1400930
weighted avg       0.71      0.71      0.71   1400930


Accuracy :  71.16244209203886


ROC_AUC :  77.71164554475179
```

We can dive deeper by illustrating the confusion matrix. From the graph below, we know that our misclassification rate is 28.82%, which means that the probability that we classify mistakenly is 0.2882.



7. K-Nearest-Neighbor

8.1 Principle

K-Nearest-Neighbor is a classification model that given a query point x0, we find the k training points x(r), r = 1,...,k closest in distance to x0, and then classify using majority vote among the k neighbors.

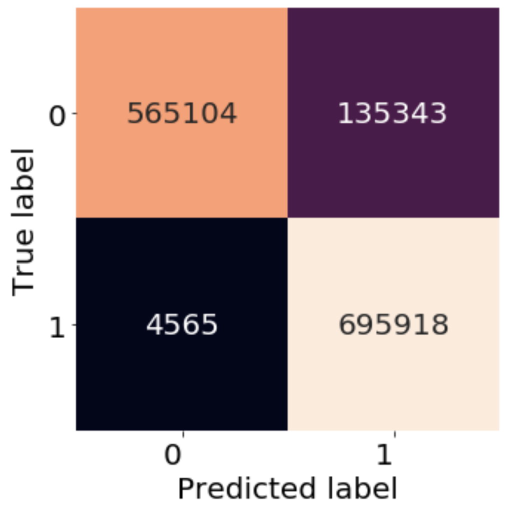
8.2 Develop KNN

Similarly, we first split the data into train and test datasets and then train our model with hyperparameter k = 3.

8.3 Model Evaluation

Both classification report and confusion matrix show that KNN classifier performs better than the logistic model. The accuracy increases to 90.01% and the misclassification rate decreases to 9.99%, which means the classifier can correctly classify 90.01% of the test dataset.

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.81      0.89    700447
         1.0       0.84      0.99      0.91    700483

    accuracy                           0.90   1400930
   macro avg       0.91      0.90      0.90   1400930
weighted avg       0.91      0.90      0.90   1400930


Accuracy :  90.01320551348033


ROC_AUC :  99.97228979749896
```



8. Random Forest

9.1 Principle

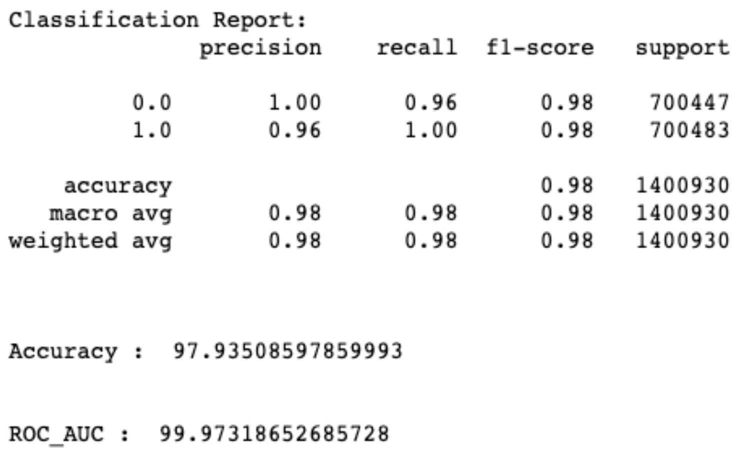
Random forest is an ensemble of decision trees and uses majority vote for classification. It is intuitive that by choosing the majority, we can get a more robust result and higher accuracy.

9.2 Develop Random Forest Model

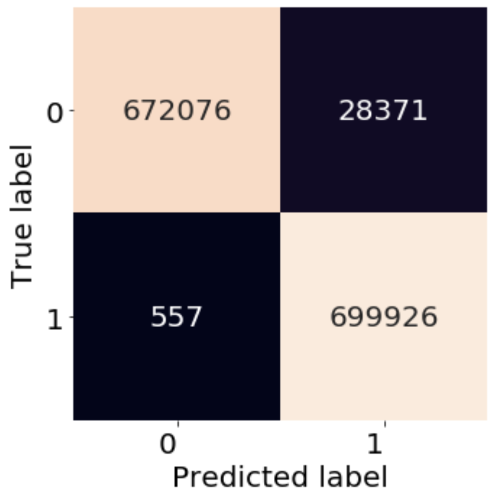
Similarly, we split the train and test datasets and then fit the model with train data. We use default value for parameters in the 'RandomForestClassifier' function.

9.3 Model Evaluation

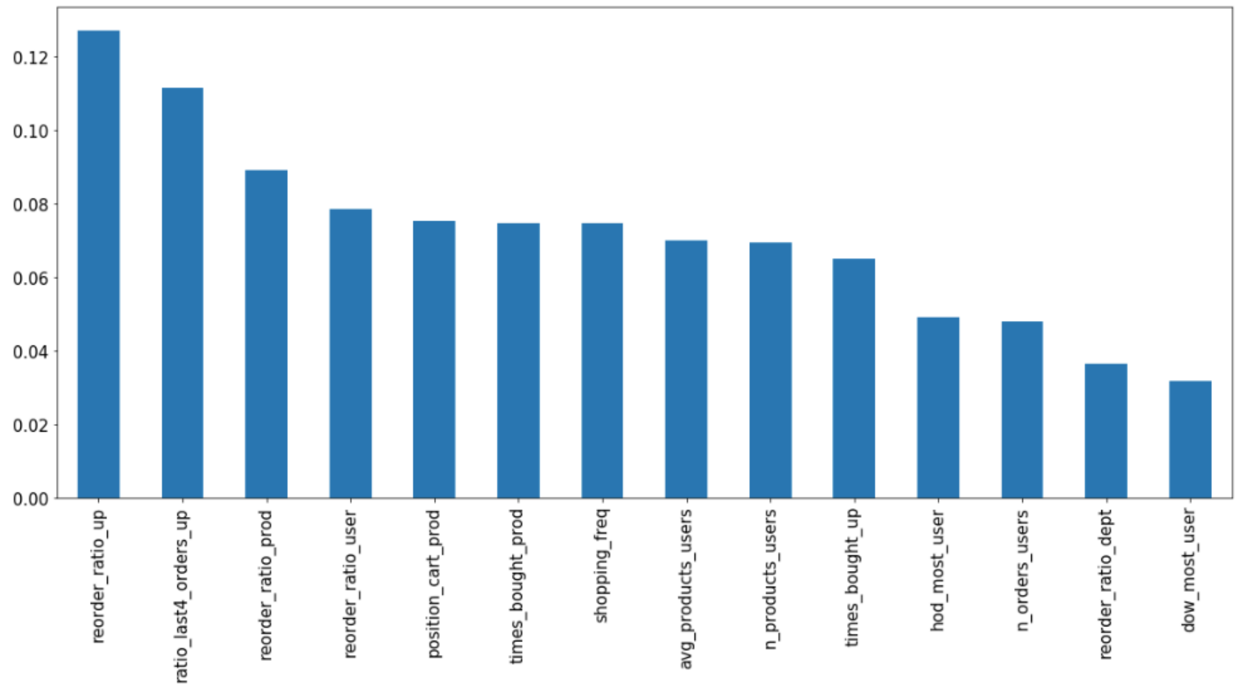
We can see from the classification report and confusion matrix below that the model performs excellent in the test dataset. The accuracy is 97.94% and the misclassification rate is 2.06%, which means the model can successfully predict 97.94% of the test dataset.



```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98    700447
         1.0       0.96      1.00      0.98    700483

    accuracy                           0.98   1400930
   macro avg       0.98      0.98      0.98   1400930
weighted avg       0.98      0.98      0.98   1400930


Accuracy :  97.93508597859993


ROC_AUC :  99.97318652685728
```



Also, we plot feature importance distribution to illustrate which feature play an crucial role in the modeling. Here the most influential feature is ‘reorder_ratio_up’, which illustrates how often the user buy the product again after first purchase. The formula we use to calculate this feature is as follows: reorder_ratio_up = times a user buys a product/number of orders after the user buy the product for the first time. Similarly, feature ‘Ratio_last4_orders_up’ describes the users’ recent purchase preference by only considering the last four previous orders. On the other hand, feature ‘dow_most_user’ which illustrates the day in a week when a user tends to purchase has the least impact on the model.



9. Summary and Conclusions

Apart from the models we discuss before, we also develop other classification models, including decision tree, SVM, and Naïve Bayes. However, due to the limitation of our computers and the big size of data, SVM takes too long to run out. Naïve Bayes performs poorly in this case, and the performance of decision tree is similar to our baseline model. So, we do not discuss in details in terms of these models.

Our goal of this project is to predict whether a user will buy a product in the next order. First, we get familiar with the data through exploratory data analysis, and then we abstract 14 features from prior orders. We develop and compare different classification models and choose random forest as our final model with 97.94% accuracy. Also, we visualize the feature importance distribution to show how features influence the model separately.

We gain some practical experience during this project. We find that SVM may not be a good classifier for big data because it would take too long to see the result. On contrast, random forest performs best in this case, and it can also show the feature importance, which many other classifiers lack.

Although the accuracy is pretty good, there are still some improvements we can make to make a better model. Here we use default value to develop random forest. Specifically, the number of decision trees in the random forest we use is the default value 100. For improvement, we can do hyperparameter tuning to find the optimal number of decision trees in our case. Besides, we use bagging to ensemble the decision tress. For improvement, we may also consider gradient boosting to ensemble decision trees.

10. References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. No. 10. New York: Springer series in statistics, 2001.