Store Management for online shopping

I Problem:

Whether you're shopping from a well-planned grocery store or guided by whimsical grazing, our unique eating habits define who we are. When buying food, we often choose to buy back the goods they need. This is an accurate recommendation model that can help users quickly find their needs.

Recently, Instacart open sourced 3 million Instacart orders. We hope to analyze the transaction data from these orders to develop models and predict the products that users will buy again.

II Goals:

As the trendy of benefit maximization, store management should be optimized with machine learning. We choose history data information about Orders, Products, Hour of a Day, The Day of a Week and so forth as the dataset, to analyze and predict two things to help reducing the cost of managing a store.

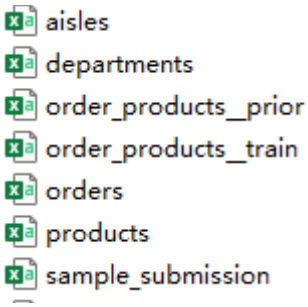
Specifically, we set two goals to do this.

1) Predict the product that the user would buy again in the next purchase.
2) According to dataset we can predict in one week which day is the most busy and which day order number is the least. Accordingly, store supervisor would like to arrange schedule more reasonable according to this prediction

III DataSet:

The dataset is a relational set of files describing customers' orders over time. The goal of the competition is to predict which products will be in a user's next order. The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, we provide between 4 and 100 of their orders, with the sequence of products purchased in each order. We also provide the week and hour of day the order was placed, and a relative measure of time between orders.

The data are given as CSV format, and the data amount is up to 200M.




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

orders.csv:

This file tells to which set (prior, train, test) an order belongs. You are predicting reordered items only for the test set orders. 'order_dow' is the day of week.

```
orders_df.head()
```

	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

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. You may predict an explicit 'None' value for orders with no reordered items.

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0

IV. Implementation

As mentioned above, we have used multiple data sets to advance and test our projects. Not only do we need to train our machine learning models through data sets, but we also need to verify our accuracy and models through data validation. Next, I will introduce the process of verifying the model accuracy using LightGBM and Xgboost algorithm implementation process.

4.1 Data Processing

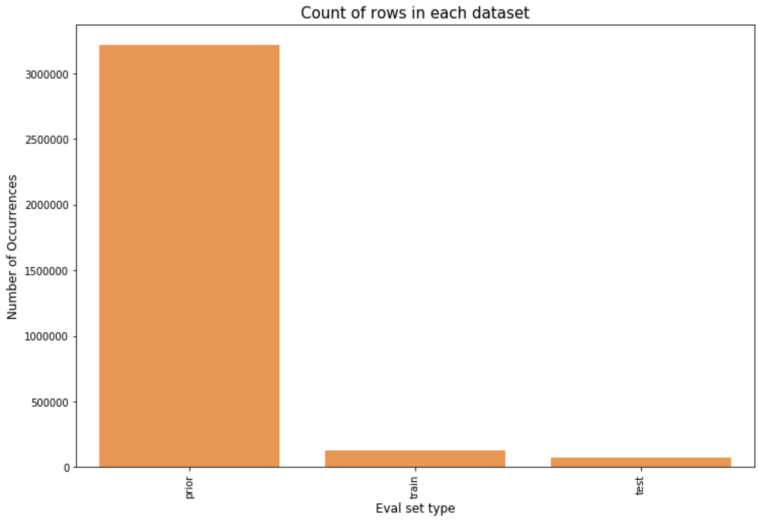
4.1.1 Goal 1

As mentioned earlier, in this dataset, 4 to 100 orders of a customer are given (we will look at this later) and we need to predict the products that will be re-ordered. So, the last order of the user has been taken out and divided into train and test sets. All the prior order information of the customer is present in order_products_prior file. We can also note that there is a column in orders.csv file called eval_set which tells us as to which of the three datasets (prior, train or test) the given row goes to.

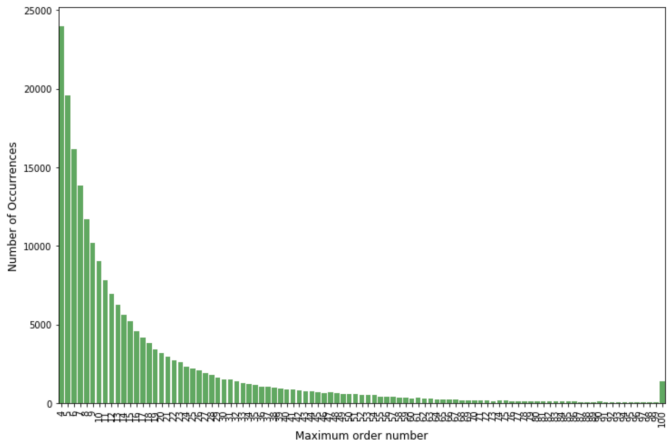
There is no NAN value in any table except the column orders['days_since_prior_order'].
And the NAN value in orders['days_since_prior_order'] means the order is the first order for that user so it doesn't have prior order. Therefore, there is no need for us to remove any row with NAN value.

Order_products*csv file has more detailed information about the products that been bought in the given order along with the re-ordered status.

1) Let us first get the count of rows in each of the three sets.

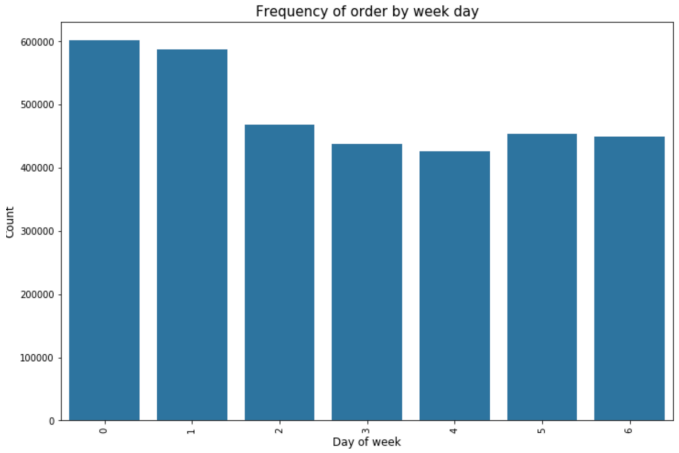


2) Now let us validate the claim that 4 to 100 orders of a customer are given.



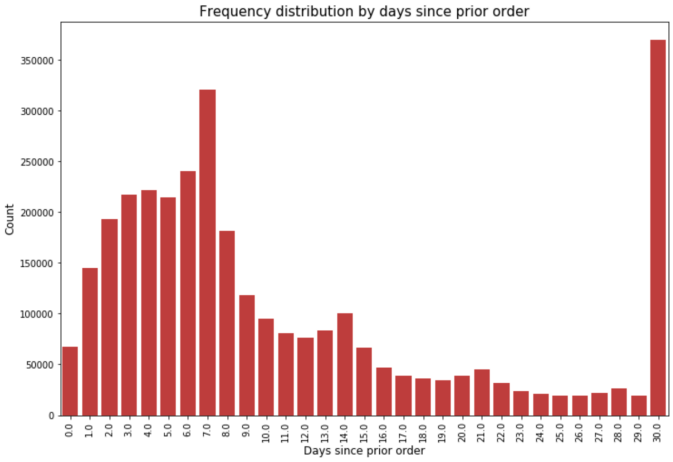
So, there are no orders less than 4 and is max capped at 100 as given in the data page.

3) Now let us see how the ordering habit changes with day of week.



Seems like 0 and 1 is Saturday and Sunday when the orders are high and low during Wednesday.

4) Now let us check the time interval between the orders.



Looks like 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).

5) Since our objective is to figure out the re-orders, let us check out the re-order percentage in prior set and train set.

```
# percentage of re-orders in prior set #
order_products_prior_df.reordered.sum() / order_products_prior_df.shape[0]
```

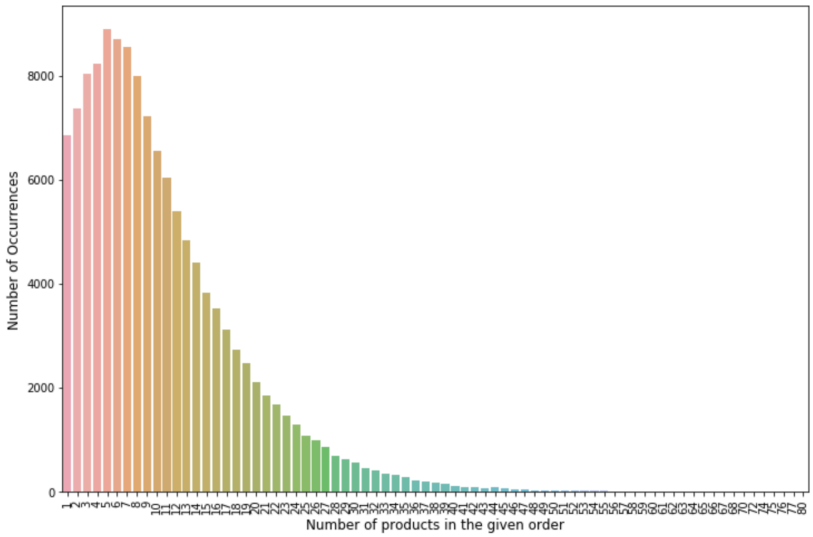
```
0.5896974667922161
```

```
# percentage of re-orders in train set #
order_products_train_df.reordered.sum() / order_products_train_df.shape[0]
```

```
0.5985944127509629
```

On an average, about 59% of the products in an order are re-ordered products.

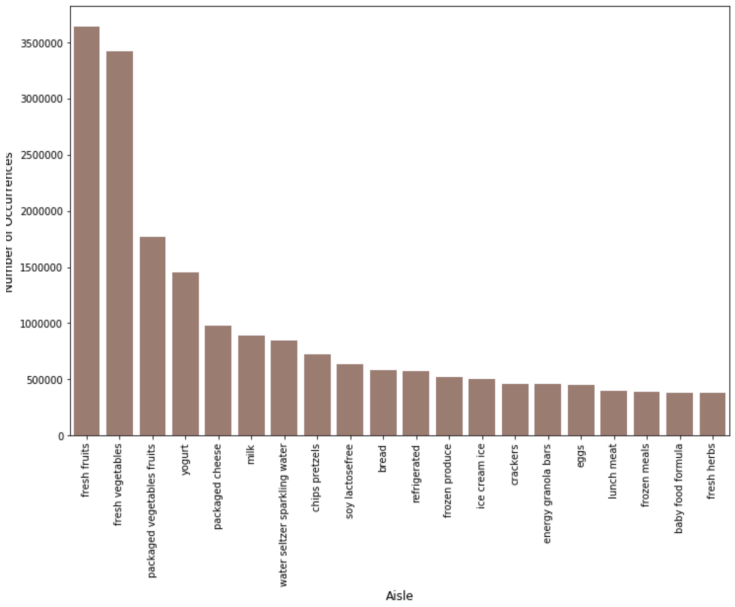
6) Now let us see the number of products bought in each order.



A right tailed distribution with the maximum value at 5.

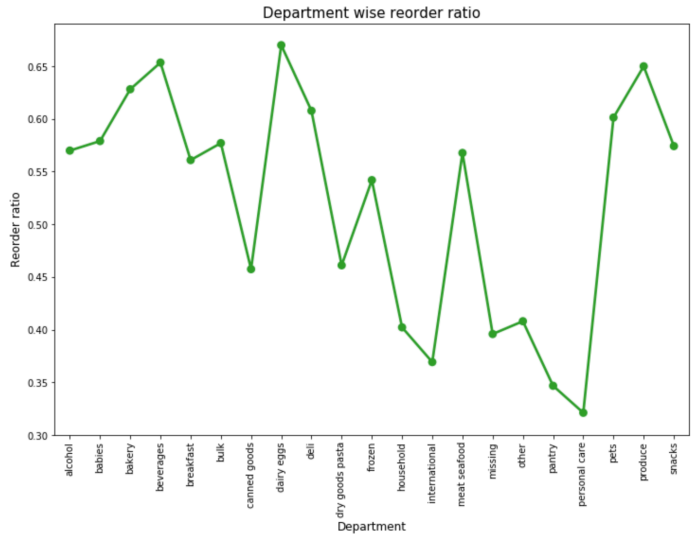
7) Now let us merge these product details with the order_prior details and look at the important aisles.

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



The top two aisles are fresh fruits and fresh vegetables.

8) Now let us check the reordered percentage of each department.



Personal care has lowest reorder ratio and dairy eggs have highest reorder ratio.

4.1.1 Goal 2

4.2 Implement of Classifications

1. LightGBM

LightGBM is a gradient boosting algorithm framework based on decision tree developed by Microsoft. It is designed to be distributed and efficient with the following advantages:

a) Faster training speed and higher efficiency.
b) Lower memory usage.
c) Better accuracy.
d) Support of parallel and GPU learning.
e) Capable of handling large-scale data.

The base idea of LightGBM is still decision tree while it improves the accuracy and efficiency.

The prediction of user's purchase is based on different factors, including

a) User related features
b) Order related features
c) Product related features
d) Users&product features

In that case, I simply reorganized the prior data framework according to the above groups, and get the light GBM dataset through that.

Redefine dataframe

```
print('user related features')
df['user_id'] = df.order_id.map(orders.user_id)
df['user_total_orders'] = df.user_id.map(users.nb_orders)
df['user_total_items'] = df.user_id.map(users.total_items)
df['total_distinct_items'] = df.user_id.map(users.total_distinct_items)
df['user_average_days_between_orders'] = df.user_id.map(users.average_days_between_orders)
df['user_average_basket'] =  df.user_id.map(users.average_basket)

print('order related features')
# df['dow'] = df.order_id.map(orders.order_dow)
df['order_hour_of_day'] = df.order_id.map(orders.order_hour_of_day)
df['days_since_prior_order'] = df.order_id.map(orders.days_since_prior_order)
df['days_since_ratio'] = df.days_since_prior_order / df.user_average_days_between_orders

print('product related features')
df['aisle_id'] = df.product_id.map(products.aisle_id)
df['department_id'] = df.product_id.map(products.department_id)
df['product_orders'] = df.product_id.map(products.orders).astype(np.int32)
df['product_reorders'] = df.product_id.map(products.reorders)
df['product_reorder_rate'] = df.product_id.map(products.reorder_rate)

print('user_X_product related features')
df['z'] = df.user_id * 100000 + df.product_id
df.drop(['user_id'], axis=1, inplace=True)
df['UP_orders'] = df.z.map(userXproduct.nb_orders)
df['UP_orders_ratio'] = (df.UP_orders / df.user_total_orders).astype(np.float32)
df['UP_last_order_id'] = df.z.map(userXproduct.last_order_id)
df['UP_average_pos_in_cart'] = (df.z.map(userXproduct.sum_pos_in_cart) / df.UP_orders).astype(np.float32)
df['UP_reorder_rate'] = (df.UP_orders / df.user_total_orders).astype(np.float32)
df['UP_orders_since_last'] = df.user_total_orders - df.UP_last_order_id.map(orders.order_number)
df['UP_delta_hour_vs_last'] = abs(df.order_hour_of_day - df.UP_last_order_id.map(orders.order_hour_of_day)).map(lambda x: min(x, 24-x)).astype(np.int8)
#df['UP_same_dow_as_last_order'] = df.UP_last_order_id.map(orders.order_dow) == \
#                                  df.order_id.map(orders.order_dow)
```

Get dataset and train the model

Train the LGB model

```
d_train = lgb.Dataset(df_train[f_to_use],
                      label=labels,
                      categorical_feature=['aisle_id', 'department_id'])  # , 'order_hour_of_day', 'dow'
del df_train
```

```
d_train
```

```
<lightgbm.basic.Dataset at 0x25e4dae96d8>
```

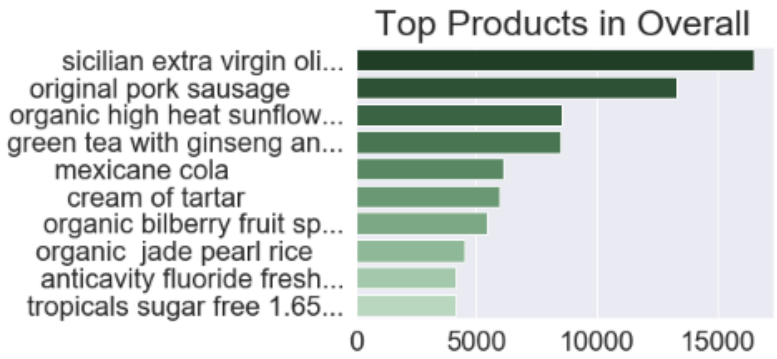
```
params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': {'binary_logloss'},
    'num_leaves': 96,
    'max_depth': 10,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.95,
    'bagging_freq': 5
}
ROUNDS = 100

print('light GBM train :-)')
bst = lgb.train(params, d_train, ROUNDS)
# lgb.plot_importance(bst, figsize=(9,20))
del d_train
```

```
light GBM train :-)
D:\Software_2\Anaconda\lib\site-packages\lightgbm\basic.py:1243: UserWarning: Using categorical_feature in Dataset.
  warnings.warn('Using categorical_feature in Dataset.')
```

Predict the users' next purchase according to Test dataset

```
### build candidates list for test ###

df_test, _ = features(test_orders)

print('light GBM predict')
preds = bst.predict(df_test[f_to_use])

df_test['pred'] = preds

TRESHOLD = 0.22  # guess, should be tuned with crossval on a subset of train data

d = dict()
for row in df_test.itertuples():
    if row.pred > TRESHOLD:
        try:
            d[row.order_id] += ' ' + str(row.product_id)
        except:
            d[row.order_id] = str(row.product_id)

for order in test_orders.order_id:
    if order not in d:
        d[order] = 'None'

sub = pd.DataFrame.from_dict(d, orient='index')
```

```
build candidate list
```



2. XGboost

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data.

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

XGBoost stands for extreme Gradient Boosting.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

Gradient Boosting algorithm also called gradient boosting machine including the learning rate.

Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.

Regularized Gradient Boosting with both L1 and L2 regularization

Creating features related to the products using product_id

	product_id	prd_count_p	p_reordered_ratio
0	1	1852	0.613391
1	2	90	0.133333
2	3	277	0.732852
3	4	329	0.446809
4	5	15	0.600000

Creating user-product features

	user_id	product_id	uxp_times_bought	uxp_reordered_ratio
0	1	196	10	0.900000
1	1	10258	9	0.888889
2	1	10326	1	0.000000
3	1	12427	10	0.900000
4	1	13032	3	0.666667

Merging all the features into data DF

	user_id	product_id	uxp_times_bought	uxp_reordered_ratio	u_num_of_orders	u_avg_prd	dow_most_orders_u	hod_most_orders_u	prd_count_p	p_reordered_ratio
0	1	196	10	0.900000	10	5.9	4	7	35791	0.776480
1	1	10258	9	0.888889	10	5.9	4	7	1946	0.713772
2	1	10326	1	0.000000	10	5.9	4	7	5526	0.652009
3	1	12427	10	0.900000	10	5.9	4	7	6476	0.740735
4	1	13032	3	0.666667	10	5.9	4	7	3751	0.657158

Creating train and test dataset

```
#filling the NAN values
data_train.reordered.fillna(0, inplace=True)
```

```
#setting user_id and product_id as index.
data_train = data_train.set_index(['user_id', 'product_id'])

#deleting eval_set, order_id as they are not needed for training.
data_train.drop(['eval_set', 'order_id'], axis=1, inplace=True)
```

```
#setting user_id and product_id as index.
data_test = data_test.set_index(['user_id', 'product_id'])
```

```
data_test.head()
```

```
#merging the aisles and department ids to with the train and test data
data_train = data_train.merge(products[['product_id', 'aisle_id']], on='product_id', how='left')
data_test = data_test.merge(products[['product_id', 'aisle_id']], on='product_id', how='left')
```

```
#department
data_train = data_train.merge(products[['product_id', 'department_id']], on='product_id', how='left')
data_test = data_test.merge(products[['product_id', 'department_id']], on='product_id', how='left')
```

```
#setting user_id and product_id as index.
data_test = data_test.set_index(['user_id', 'product_id'])
#setting user_id and product_id as index.
data_train = data_train.set_index(['user_id', 'product_id'])
```

Creating Predictive model --- XGBoost

```
#Building a XGBoost model.
#importing the package.
import xgboost as xgb

#splitting the train data into training and testing set.
X_train, y_train = data_train.drop('reordered', axis=1), data_train.reordered

#setting boosters parameters
parameters = {
    'eavl_metric' : 'logloss',
    'max_depth' : 5,
    'colsample_bytree' : 0.4,
    'subsample' : 0.8
}

#instantiating the model
xgb_clf = xgb.XGBClassifier(objective='binary:logistic', parameters=parameters, num_boost_round=10)

# TRAIN MODEL
model = xgb_clf.fit(X_train, y_train)

#FEATURE IMPORTANCE - GRAPHICAL
xgb.plot_importance(model)
```

predicting on the testing data

```
#predicting on the testing data
y_pred = xgb_clf.predict(data_test).astype('int')

#setting a threshold.
y_pred = (xgb_clf.predict_proba(data_test)[:, 1] >= 0.21).astype('int')
y_pred[0:10]
```

```
#saving the prediction as a new column in data_test
data_test['prediction'] = y_pred
data_test.head()
```

```
# Reset the index
final = data_test.reset_index()
# Keep only the required columns to create our submission file (for chapter 6)
final = final[['product_id', 'user_id', 'prediction']]

gc.collect()
final.head()
```

	product_id	user_id	prediction
0	248	3	0
1	1005	3	0
2	1819	3	0
3	7503	3	0
4	8021	3	0

```
#Creating a submission file
orders = pd.read_csv('orders.csv')
orders_test = orders.loc[orders.eval_set == 'test', ['user_id', 'order_id']]
orders_test.head()
```

	user_id	order_id
38	3	2774568
44	4	329954
53	6	1528013
96	11	1376945
102	12	1356845

```
#merging our prediction with orders_test
final = final.merge(orders_test, on='user_id', how='left')
final.head()
```

	product_id	user_id	prediction	order_id
0	248	3	0	2774568
1	1005	3	0	2774568
2	1819	3	0	2774568
3	7503	3	0	2774568
4	8021	3	0	2774568

We now check how the dictionary were populated (open hidden output)

```
d = dict()
for row in final.itertuples():
    if row.prediction== 1:
        try:
            d[row.order_id] += ' ' + str(row.product_id)
        except:
            d[row.order_id] = str(row.product_id)

for order in final.order_id:
    if order not in d:
        d[order] = 'None'

gc.collect()
```

```
#Convert the dictionary into a DataFrame
sub = pd.DataFrame.from_dict(d, orient='index')

#Reset index
sub.reset_index(inplace=True)
#Set column names
sub.columns = ['order_id', 'products']

sub.head()
```

	order_id	products
0	2774568	9387 17668 21903 22035 39190 43961 47766
1	1528013	21903 38293
2	1376945	8309 8670 14947 27959 28465 33572 34658 35640 ...
3	1356845	7076 10863 13176 14992 21616 28134
4	2161313	196 10441 11266 12427 14715 27839 37710

3. Linear Regression

1) **Linear Regression**

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output).

Linear regression Model is very easy but very precise and explainable. Each data owns n features and each feature has its own weight. Adding an offset value to multiply of weight and feature is the model.

$$f(x) = w_1x_1 + w_2x_2 + ... + w_dx_d + b$$

2) Order Data

After reading the data dealt above, ascending ordering all the data according to the date with sort_values method.

```
#根据order_date来排序
orders = orders.sort_values('order_date', inplace=False)
orders
```

	user_id	order_dow	order_hour_of_day	days_since_prior_order	order_date	product_count
3058067	188506	5	16	10.0	20160101	15
2443702	150384	5	9	-1.0	20160101	6
2443344	150361	5	10	6.0	20160101	12
1604966	98519	5	19	3.0	20160101	11
2442300	150307	5	10	7.0	20160101	14
2442093	150299	5	14	12.0	20160101	15
2441829	150285	5	13	3.0	20160101	15
1604767	98514	5	16	1.0	20160101	8
1605659	98548	5	20	7.0	20160101	15
2440578	150208	5	10	-1.0	20160101	16
520720	32099	5	13	1.0	20160101	5

3) Storing ‘order_dow’ in advance

```
dow =[5]
od_record = 5
for orders_dow in orders['order_dow']:
    if orders_dow == od_record:
        continue
    else:
        dow.append(orders_dow)
    od_record = orders_dow
dow
```

4) Count the total product counts of each day in order to observe the relationship between product counts per day and specific day of now week. And dealing the ‘order_dow’ column.

```
#根据order_date将多行数据统一
orders = orders.groupby('order_date').sum()
```

order_date	user_id	order_dow	order_hour_of_day	days_since_prior_order	product_count
20160101	637733110	32880	87990	68433.0	66793
20160102	664002613	41184	92502	69461.0	69348
20160103	704973498	50743	97874	73967.0	74928
20160104	729899306	7525	100688	77540.0	77118
20160105	744965569	15506	104163	80287.0	77458
20160106	804360145	24885	112052	85469.0	84157
20160107	855453642	35156	118437	90039.0	89620
20160108	901595980	47055	127208	95997.0	95676
20160109	968755114	60114	134916	102479.0	102387
20160110	997754770	72331	139628	106150.0	103378

```
orders['order_dow']=dow
orders
```

order_date	user_id	order_dow	order_hour_of_day	days_since_prior_order	product_count
20160101	637733110	5	87990	68433.0	66793
20160102	664002613	6	92502	69461.0	69348
20160103	704973498	7	97874	73967.0	74928
20160104	729899306	1	100688	77540.0	77118
20160105	744965569	2	104163	80287.0	77458
20160106	804360145	3	112052	85469.0	84157
20160107	855453642	4	118437	90039.0	89620
20160108	901595980	5	127208	95997.0	95676
20160109	968755114	6	134916	102479.0	102387
20160110	997754770	7	139628	106150.0	103378
20160111	1043985705	1	144820	110790.0	109288
20160112	1096337176	2	150244	113187.0	113346
20160113	1122575392	3	155665	120073.0	117058
20160114	1161204065	4	160120	122629.0	121649
20160115	1226758698	5	171510	130896.0	128428
20160116	1292675383	6	179762	138113.0	134786
20160117	1342713659	7	183951	139977.0	139466
20160118	1373164678	1	191411	146728.0	144523
20160119	1416649198	2	197915	149349.0	147675
20160120	1463161482	3	201754	154342.0	152891

5) Select X and Y

```
import math
X=orders.drop(['product_count', 'order_hour_of_day', 'user_id'],axis=1)
Y=orders['product_count']
```

6) PCA, Standard data

:
```
from sklearn.decomposition import PCA
transfer = PCA(n_components=0.90)
xT = transfer.fit_transform(X)
```

:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=22)
```

:
```
# Standard the data using standardScalar
from sklearn.preprocessing import StandardScaler
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
x_test = transfer.transform(x_test)
```

7) Select estimator and predict which day is most busy

```
# Standard the data using standardScalar
from sklearn.preprocessing import StandardScaler
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
x_test = transfer.transform(x_test)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression, SGDRegressor
```

```
estimator = LinearRegression()
estimator.fit(x_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
print("coef:", estimator.coef_)
print("intercept:", estimator.intercept_)
```

```
coef: [-30.65446151   0.98744237]
intercept: 197.77441967846244
```

```
y_predict = estimator.predict(x_test)
print(y_predict)
```

V. Results

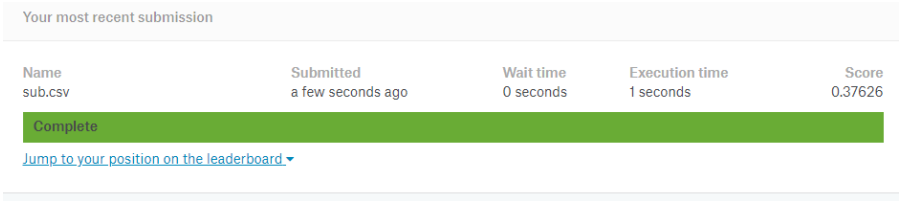
5.1 Goal_1

1) LightBGM

The final results are output as CSV format and after submission in Kaggle, the f1 score is 0.37626.

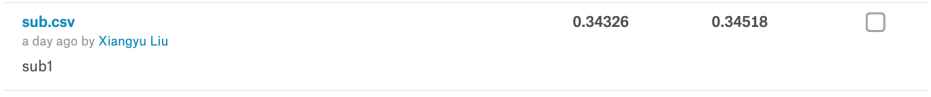
	order_id	products
0	2774568	17668 21903 39190 47766 18599 43961 23650 24810
1	1528013	21903 38293
2	1376945	33572 28465 27959 44632 24799 34658 14947 8309...
3	1356845	11520 14992 7076 28134 10863 21616 13176
4	2161313	11266 196 10441 12427 37710 48142 14715 27839
5	1416320	5134 21903 21137 24852 17948 41950 21616 24561
6	1735923	17008 2192 196 15599 31487 15131 35123 12108 3...
7	1980631	13575 6184 9387 46061 13914 41400 22362
8	139655	27845
9	1411408	26452 22008
10	2940603	30592 19894 44632 10339 14947 18531 31615
11	1192143	47626 27307 24852
12	280888	32566
13	3202221	49215 10831 21137 4793 17630 24852 45364 13629...
14	3222866	40706 13187 37131 32912 7969 38690 33198 8501 ...
15	707453	45066 42585 48230 44142 21137 47766 694 18150 ...

The top 10 popular products according to the prediction

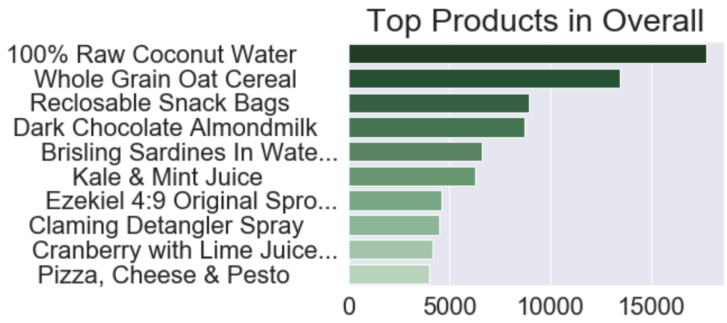


2) XGboost

The final results are output as CSV format and after submission in Kaggle, the f1 score is 0.34518.



The top 10 popular products according to the prediction



Conclusion

After comparison, Light GBM method is more accurate which has score of 0.37625 while the XGboost got 0.34518. Therefore, Light GBM would be the optimal for this problem.

5.2 Goal 2

1) Linear Regression

Using Linear Regression Score Method, R2 Score Method, Square Method, and MSE Method(mean squared error) to score the linear regression. The result is 0.99 and 626213.7.

```
score = estimator.score(x_test, y_test)
score
```

```
0.9997392932251432
```

```
from sklearn import metrics
r2S = metrics.r2_score(y_test,y_predict)
r2S
```

```
0.9997392932251432
```

```
import numpy as np
MSE = np.square(np.subtract(y_test,y_predict)).mean()
MSE
```

```
626213.727816968
```

```
MSE2 = metrics.mean_squared_error(y_test,y_predict)
MSE2
```

```
626213.7278169685
```

Only select order_dow (days of one week to act as abscissa)

```
# 确定x_x里面的值只有orderdow就行
import numpy as np
x_x=x_test['order_dow']
x_x = np.array(x_x)
x_x
```

```
array([1, 4, 7, 1, 3, 7, 5, 6, 5, 6, 7, 4, 5, 2, 7, 2, 1, 2, 2, 2, 4, 5,
       6, 2, 6, 2, 6, 6, 5, 4, 4, 4, 4, 7, 1, 3, 1, 6, 7, 4, 5, 7, 5, 7,
       3, 3, 4, 6, 1, 5, 2, 6, 2, 2, 5, 4, 5, 1, 3, 3, 2, 7, 6, 6, 3, 3,
       2, 1, 3, 7, 4, 2, 2, 3, 7, 3, 6, 6, 1, 6, 3, 3, 6, 1, 7, 3, 4, 1,
       6, 7, 3, 7, 4, 5, 5, 1, 4, 2, 7, 4, 4, 2, 3, 5, 5, 2, 1, 7, 7, 6,
       5, 6, 4, 6, 3, 6, 5, 3, 1])
```

```
bar_table = pd.DataFrame({
    'order_dow':x_x,
    'product_prediction':y_predict
})
bar_table
```

	order_dow	product_prediction
0	1	89079.393474
1	4	79435.900024
2	7	40699.391963
3	1	29377.640211
4	3	59281.257514
5	7	54464.338630
6	5	156142.354630
7	6	77057.063853
8	5	161380.736414
9	6	49129.231242
10	7	31648.495179

```
bar_y_choose_prepare = bar_table.groupby('order_dow').sum()
bar_y_choose_prepare
```

	product_prediction
order_dow	
1	1.155858e+06
2	1.093093e+06
3	1.372258e+06
4	1.192093e+06
5	1.613191e+06
6	1.691443e+06
7	1.311225e+06

The relationship between each day of one week and product count is below:

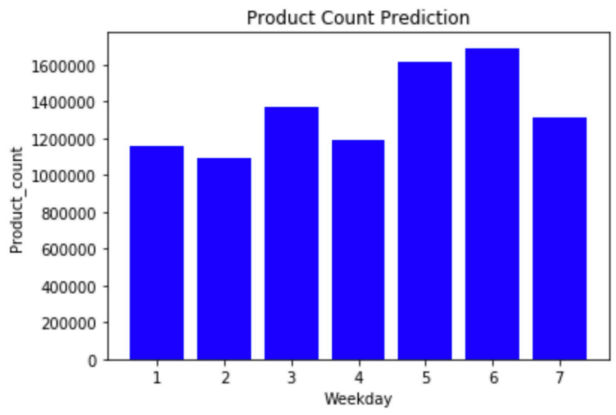
```
y_lable_value=bar_y_choose_prepare['product_prediction']
y_lable_value
```

```
order_dow
1    1.155858e+06
2    1.093093e+06
3    1.372258e+06
4    1.192093e+06
5    1.613191e+06
6    1.691443e+06
7    1.311225e+06
Name: product_prediction, dtype: float64
```

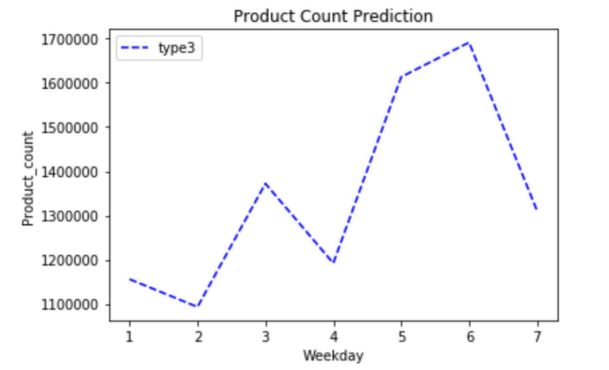
```
x_label_value=[1,2,3,4,5,6,7]
x_label_value
```

```
[1, 2, 3, 4, 5, 6, 7]
```

```
plt.bar(x_label_value, y_lable_value, fc='b')
plt.title("Product Count Prediction")
plt.xlabel("Weekday")
plt.ylabel("Product_count")
plt.show()
```




```
plt.plot(x_label_value, y_lable_value,'b--', label='type3')
plt.title('Product Count Prediction')
plt.xlabel('Weekday')
plt.ylabel('Product_count')
plt.legend()
plt.show()
```



According to the predict result, Friday and Saturday are the most busy days, and Monday and Tuesday are relative leisure time. So the store can assigned staffs optimally.