Instacart Data Analysis

~ Sanchari Chowdhuri

Contents

1. Dataset Overview

- The dataset contains order details for **206,209** Instacart Users.
- The dataset has **3,421,083** orders.
- The dataset has **49,688** product details.
- All these products are spread across**134** different aisles and belong to **21** different departments.
- For these 206209 Users, their previous order details are available as “prior”. Their latest order is segregated into training and testing order.

Relationship among different files of the dataset

```
order_products_prior_df.head(3)
```

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

```
orders_df.head()
```

order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
1	112108	train	4	4	10	9.0
2	202279	prior	3	5	9	8.0
3	205970	prior	16	5	17	12.0
4	178520	prior	36	1	9	7.0

```
products_df.head()
```

	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

```
aisles_df.head()
```

	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

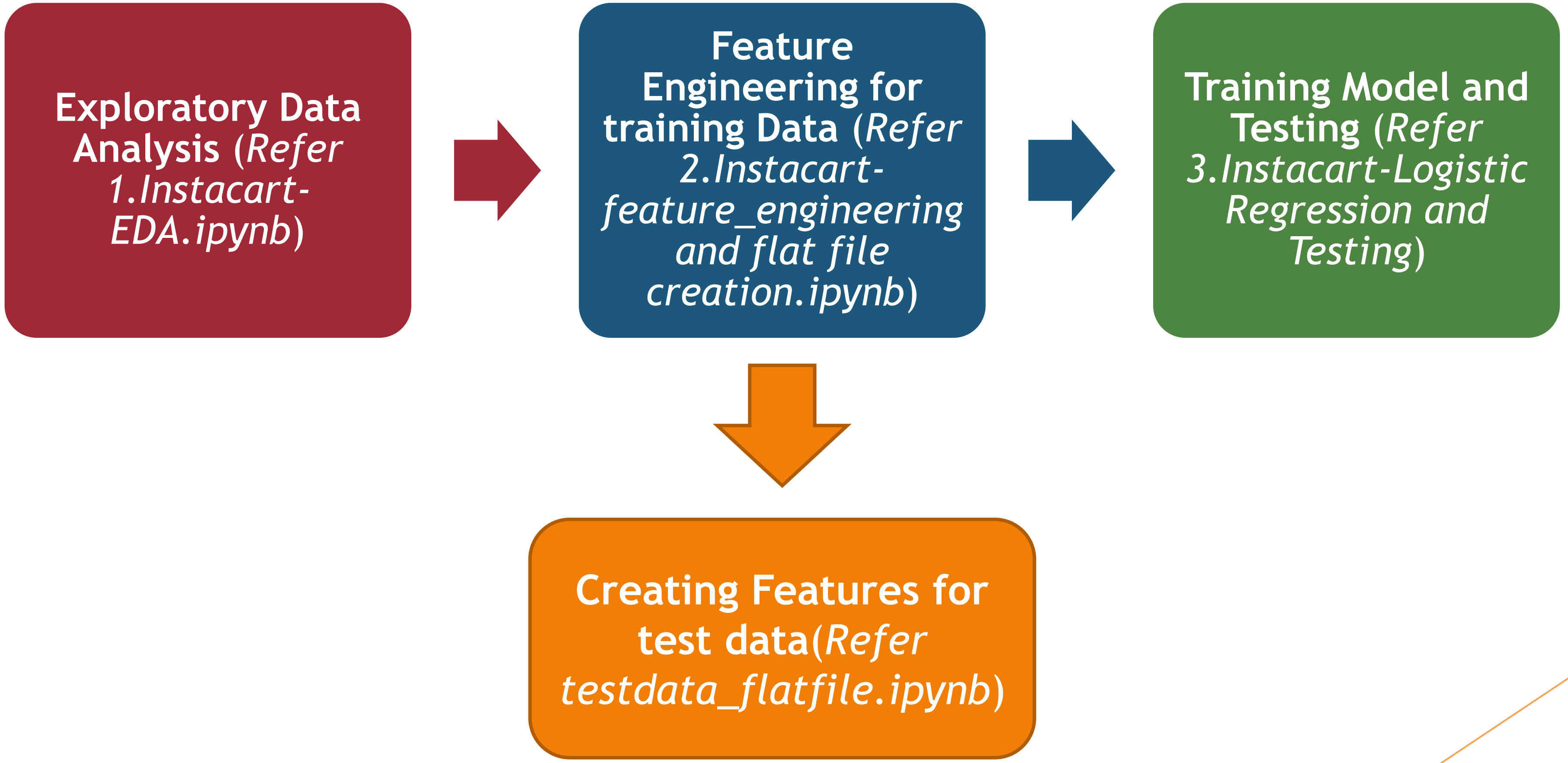
```
departments_df.head()
```

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

Scope

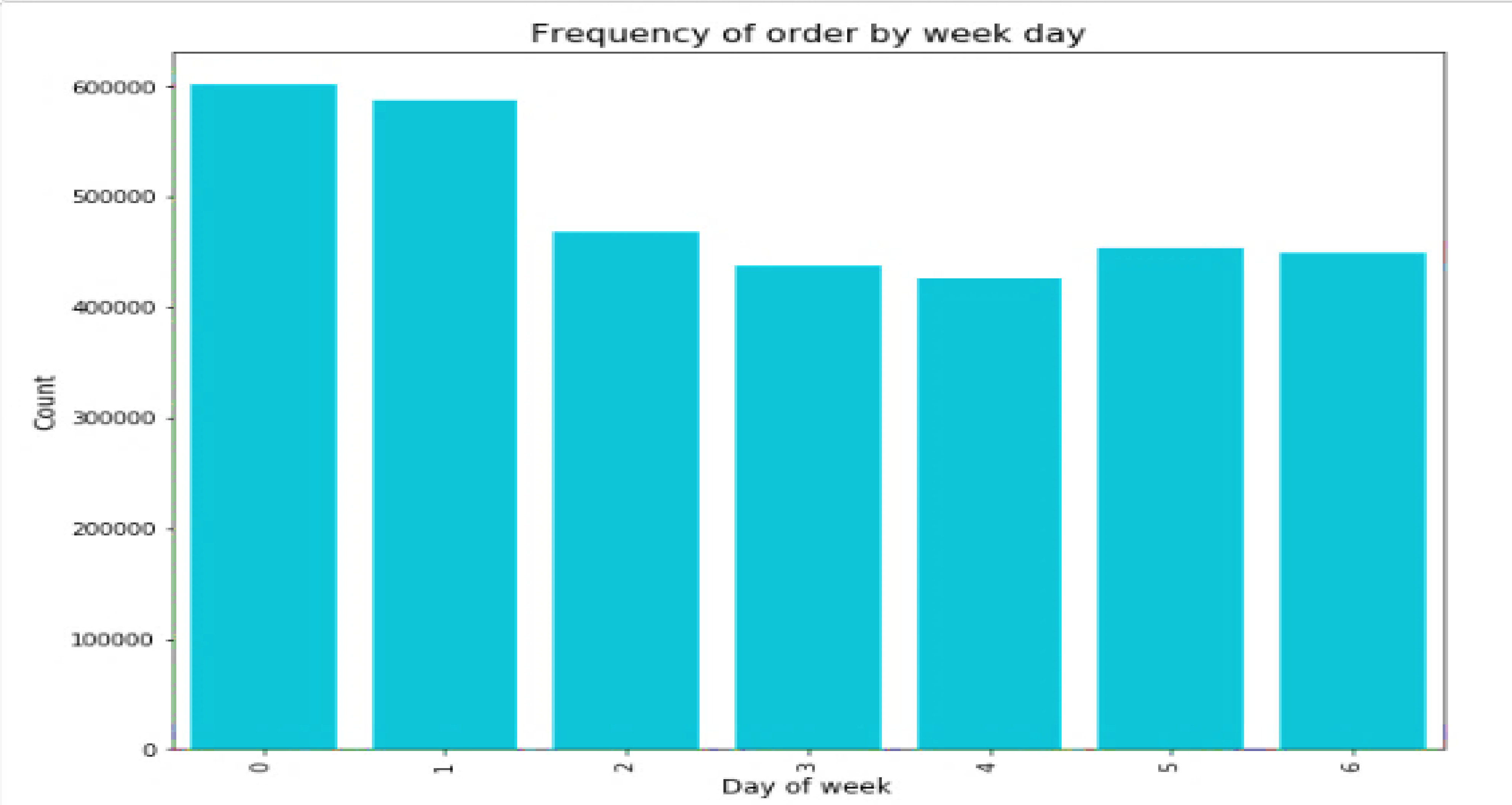
- To predict what a user might purchase in their next order.

2.Analysis Flow



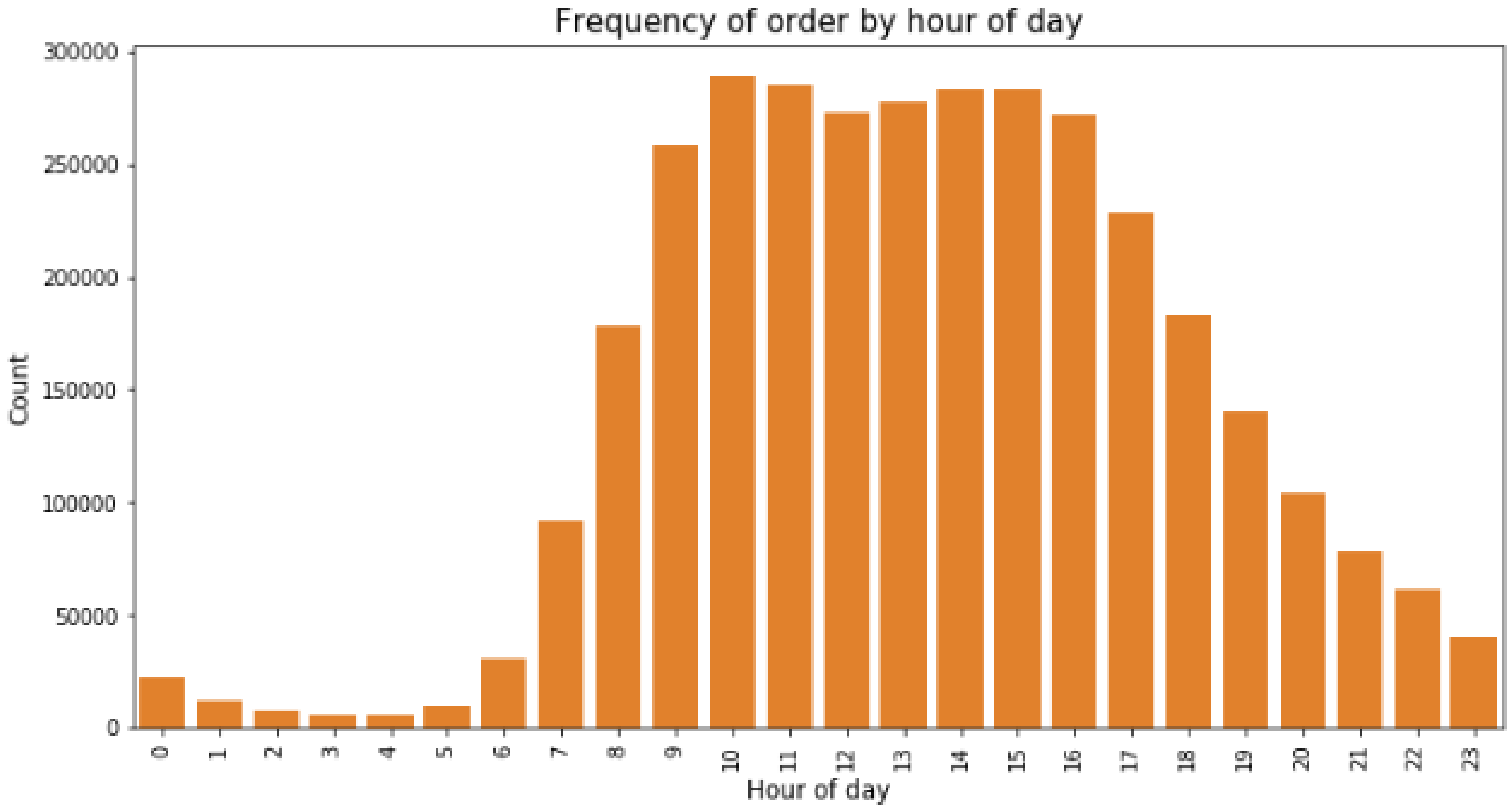
3. Exploratory Data Analysis

Number of Orders per day of week



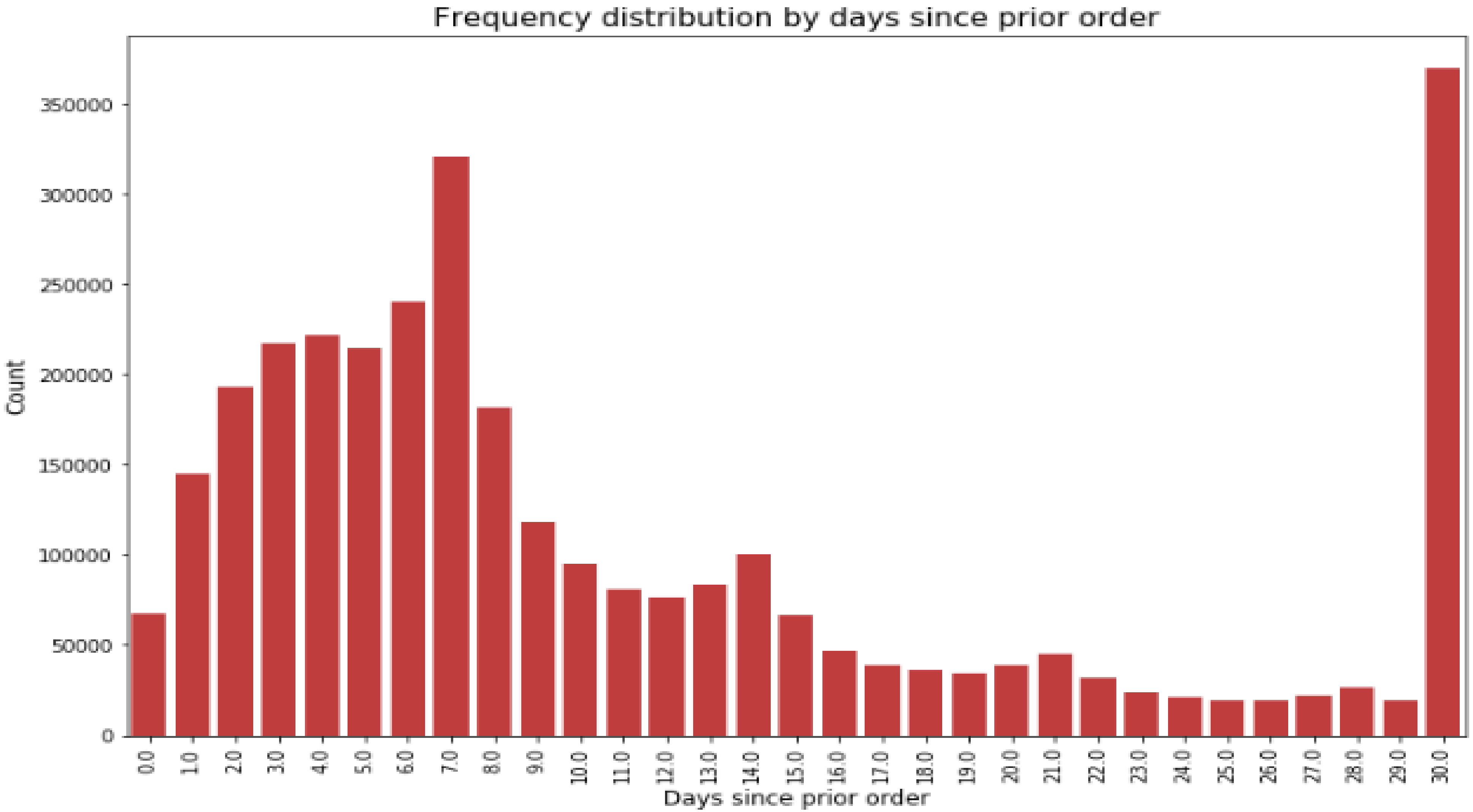
- *Sunday and Monday seems to have higher volume of orders than other days of the week*
- *Towards mid week there is a dip in order volume suggesting people prefer to get their groceries towards weekend or beginning of the week*

Orders by hours of the Day



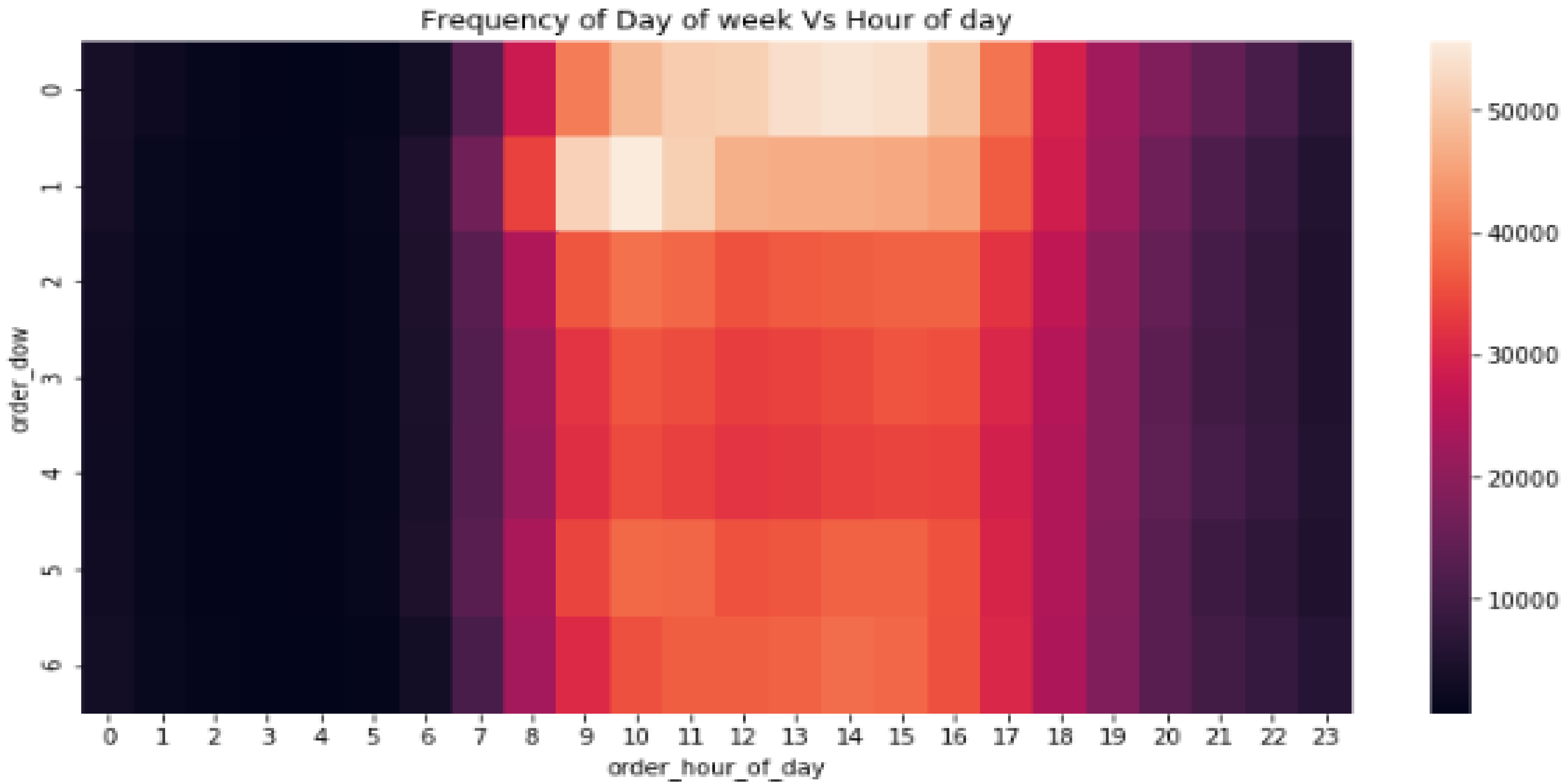
- *People order groceries in day time*

How frequently do customers order?



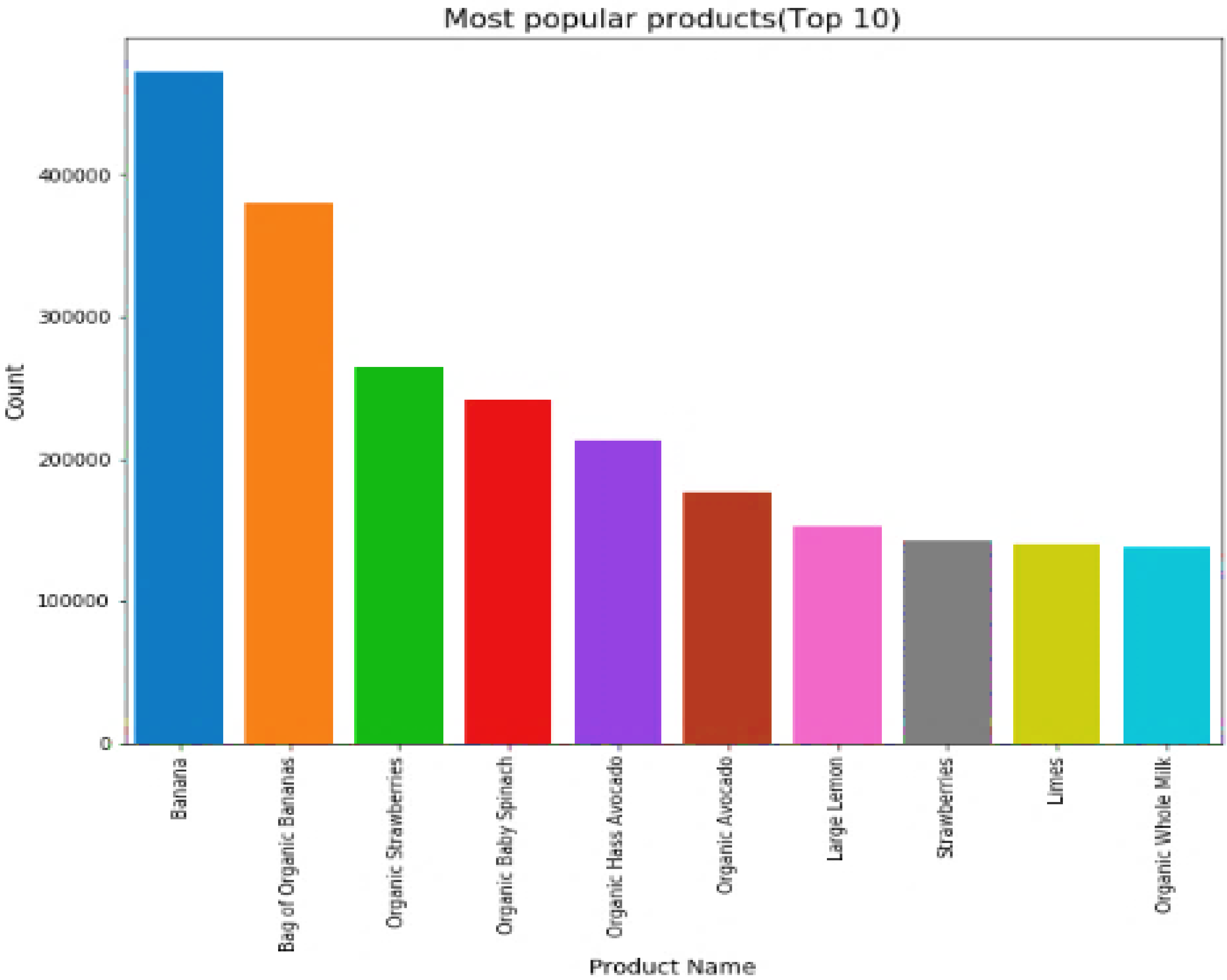
Customers order Weekly and monthly the most

When do customers order the most

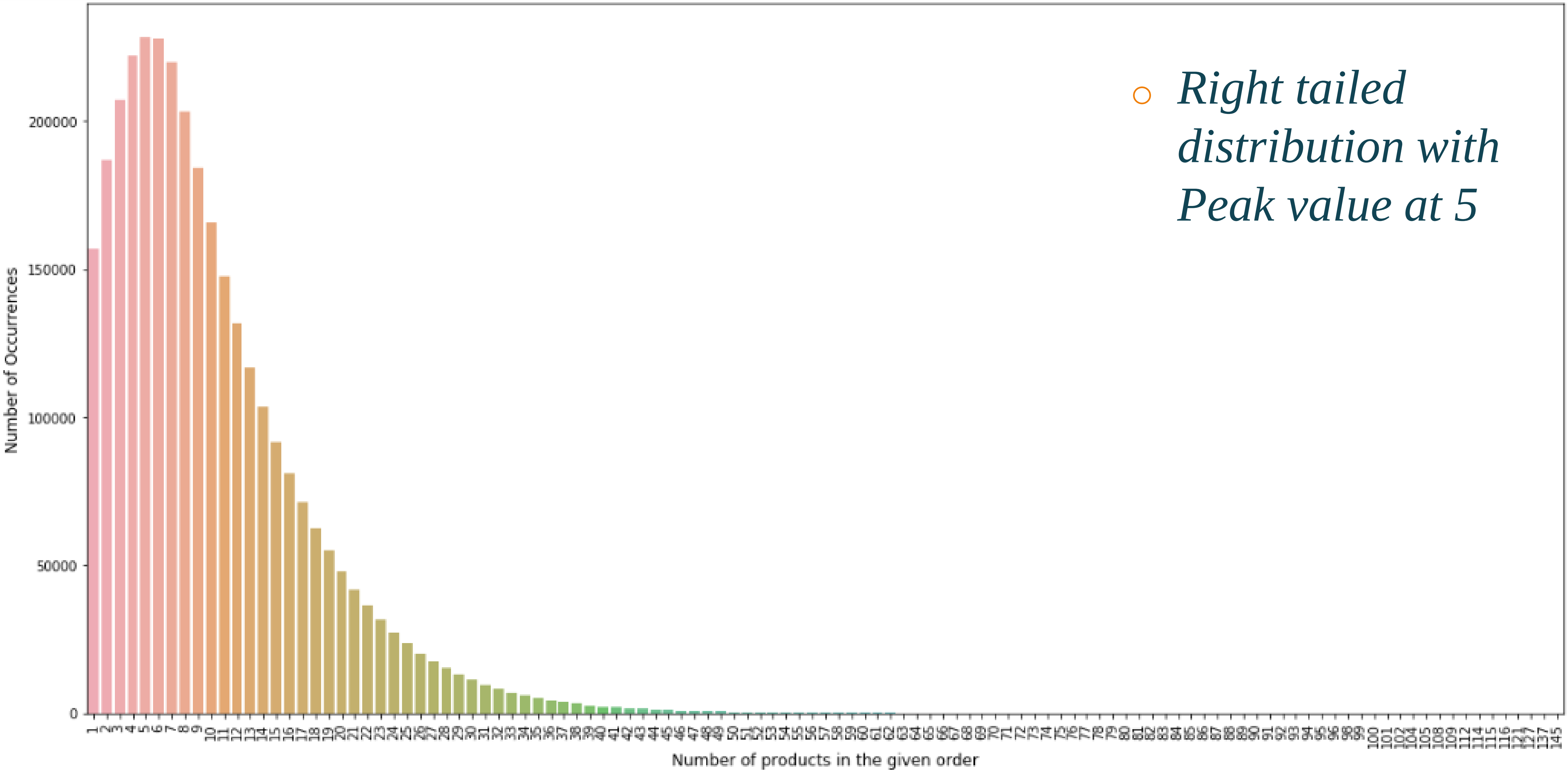


- *Heatmap showing volume of orders across hour of day and day of week*

Most Popular products

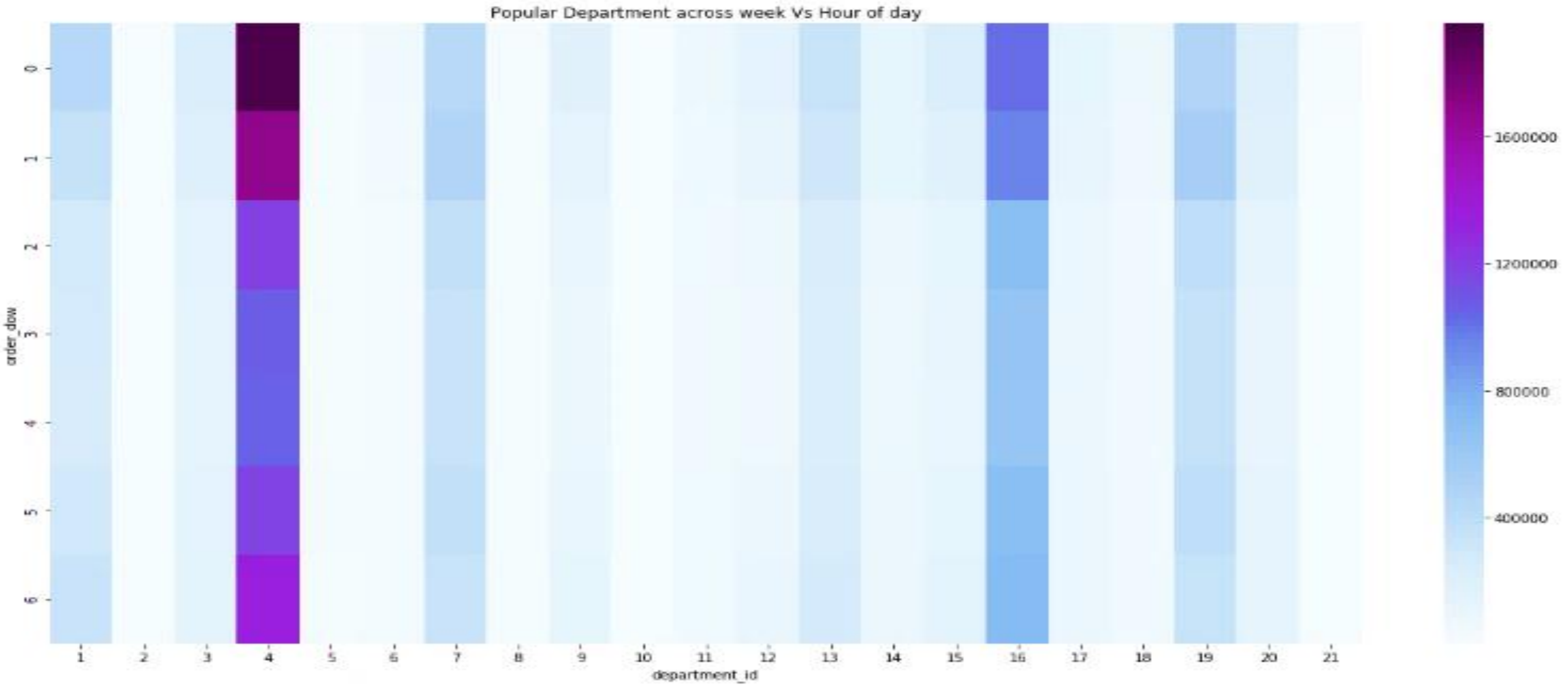


Number of Products in an Order



- *Right tailed distribution with Peak value at 5*

Popular Department across day of week



- *Department id 4 and 16 are popular throughout the week*

Department wise reorder ratio

department_id	department
1	frozen
2	other
3	bakery
4	produce
5	alcohol
6	international
7	beverages
8	pets
9	dry goods pasta
10	bulk
11	personal care
12	meat seafood
13	pantry
14	breakfast
15	canned goods
16	dairy eggs
17	household
18	babies
19	snacks
20	deli

- *Department id 4 and 16 have high reorder ratio*

4. Feature Engineering

User-Product Based Features	• user_product_avg_add_to_cart_order • user_product_total_orders • user_product_avg_days_since_prior_order • user_product_avg_order_dow • user_product_avg_order_hour_of_day
Product Based Features	• product_total_orders • product_avg_add_to_cart_order • product_avg_order_dow • product_avg_order_hour_of_day • product_avg_days_since_prior_order
User Based Feature	• user_total_orders | user_avg_cartsize • user_total_products | user_avg_days_since_prior_order • user_avg_order_dow • user_avg_order_hour_of_day • user_product_order_freq
User-Product Delta Features	• product_total_orders_delta_per_user • product_avg_add_to_cart_order_delta_per_user • product_avg_order_dow_per_user • product_avg_order_hour_of_day_per_user • product_avg_days_since_prior_order_per_user

Feature Engineering

- 23 new features are created for User- product pair

Feature Name	Description
user_product_avg_add_to_cart_order	*this column tells the average add to cart order of the product for this user*
user_product_total_orders	*how many times this product was ordered by this user*
user_product_avg_days_since_prior_order	*average number of days elapsed since last time this product was ordered by the user*
user_product_avg_order_dow	*average day of the week when the user orders this product*
user_product_avg_order_hour_of_day	*average hour of the day when the user orders this product.*
In_cart	*This tells whether a prior product ordered by the user is also present in the current order*
product_total_orders	*How many times a given product has been ordered overall*
product_avg_add_to_cart_order	*This tells the average add to cart order of the product*
product_avg_order_dow	*This tells the average day of week when this product is ordered*
product_avg_order_hour_of_day	*the average hour of the day when this product is ordered the most*
product_avg_days_since_prior_order	*average number of days elapsed since this product was last ordered*

Feature Engineering (....contd)

Feature Name	Description
user_total_orders	*Total number of orders placed by the user*
user_avg_cartsize	*Average cart size of the user*
user_total_products	*Total number of products ordered by the user*
user_avg_days_since_prior_order	*Number of days elapsed between subsequent orders*
user_avg_order_dow	*Average day of the week when user places order*
user_avg_order_hour_of_day	*Average hour of the day when user places order*
user_product_order_freq	*Ratio of user_product_total_orders and user_total_orders*
product_total_orders_delta_per_user	*difference between total number of orders placed for the product and total number of orders placed for the product by the specific user.*
product_avg_add_to_cart_order_delta_per_user	*difference between product's average add to cart order based on all users and product's average add to cart order based on this specific users.*
product_avg_order_dow_per_user	*difference between average day of week when the product is ordered based on all users and average day of week when the product is ordered based on this specific user*
product_avg_order_hour_of_day_per_user	*difference between product's average hour of day when ordered and product's average hour of day when ordered by this user*
product_avg_days_since_prior_order_per_user	*difference between product's average days elapsed since last order placed and average days elapsed since last order placed by specific user*

5.Model Creation and Performance Metrics

Training Model 1

- Model Used :Logistic Regression
- dependent variable = in_cart [0,1]
- Independent variable are all the previous mentioned features.

Performance Metrics

- Overall Model Accuracy : 85.2%

```
# Create Logistic Regression classifier object
lr = LogisticRegression(class_weight='balanced')

#Train Logistic Regression classifier
log_sm_reg=lr.fit(X_tr, y_tr)

#Predicting for test data called X_te (this is obtained by splitting 20% of train data)
y_pred_LR = log_sm_reg.predict(X_te)
scores_LR = metrics.accuracy_score(y_te, y_pred_LR)
scores_LR
```

```
0.852246300278243
```

- Although the Overall Accuracy is high but due to class imbalances accuracy is not the best metrics to quantify the classifier's performance.

- Precision, Recall and F1 Score

```
from sklearn.metrics import classification_report
print(classification_report(y_te,y_pred_LR))
```

```
             precision    recall  f1-score   support

          0       0.94      0.90      0.92   1065128
          1       0.34      0.47      0.39    121245

avg / total       0.88      0.85      0.86   1186373
```

- F1- Score Class 1 =0.39
- Precision Class1=34%
- Recall Class1 = 47%

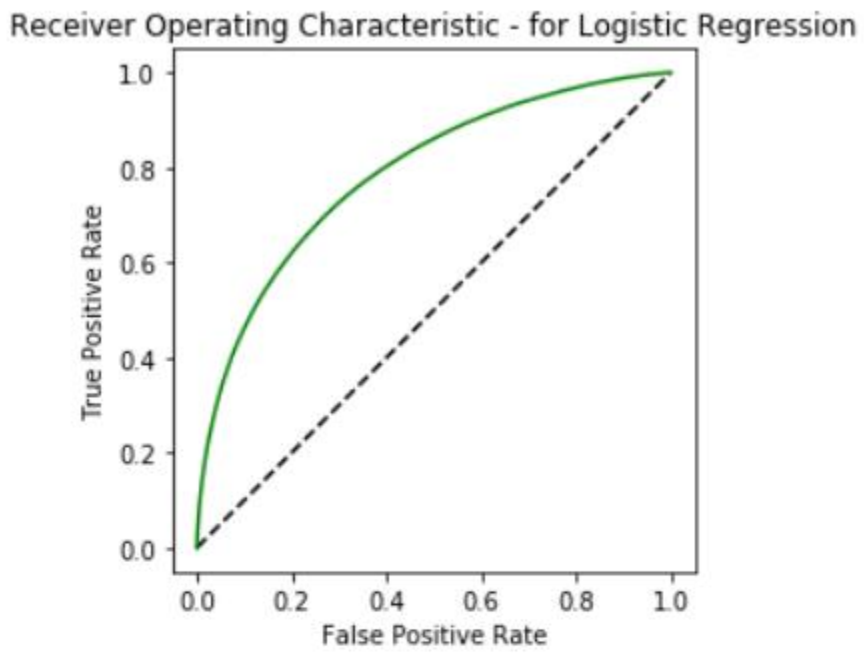
Confusion Matrix

```
pd.crosstab(y_te,y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	All
True			
0	954025	111103	1065128
1	64188	57057	121245
All	1018213	168160	1186373

- **Out of 1065128 instances of a product not being ordered -**
  - 954025 times the classifier was correctly able to predict that the product would not be reordered
  - 111103 times the classifier misclassified a not ordered product as reordered product.
- **Out of 121245 instances of a product being reordered -**
  - 64188 times the classifier misclassified a reordered product as not currently ordered.
  - 57057 times the classifier correctly classified product as reordered

- ROC Curve



0.7868745505797903

- Area Under Curve is 0.78

Test Data (*order_products__test_cap.csv*)

- The test data contains 32804 unique order ids which belongs to 32804 users.
- These are testing users.
- This test data have to be normalized (by merging prior order-product history of 32804 users)
- After all the product, user and product-user based features are obtained for these 32804 test users, the classifier trained previously will be used to predict the products ordered by these 32804 test users.

Model Metrics on Test Data.

- Model Accuracy : 76.7%
- Precision ,Recall, F1 Score

	precision	recall	f1-score	support
0	0.95	0.78	0.86	1786248
1	0.25	0.65	0.36	200771
avg / total	0.88	0.77	0.81	1987019

On the test dataset:

- **Class 0** : Precision = 95% | Recall =78% | F1 Score=0.86
- **Class 1** : Precision = 25% | Recall =65% | F1 Score=0.36

- Confusion Matrix:

Predicted	**0**	**1**	**All**
True			
0	1396049	390199	1786248
1	70991	129780	200771
All	1467040	519979	1987019

Training Model 2

- Model Used :**Random Forest**
- Dependent variable = in_cart [0,1]
- Independent variable are all the previous mentioned features.

Performance Metrics

- Overall Model Accuracy : 91.12%

```
from sklearn.ensemble import RandomForestClassifier

# Create the model with 100 trees
RandomForest = RandomForestClassifier(n_estimators=100, random_state=50, max_features = 'sqrt',n_jobs=-1, verbose = 1)

# Fit on training data
Random_Forest=RandomForest.fit(X_tr, y_tr)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 26.7min finished
```

```
#Predicting for test data called X_te (this is obtained by splitting 20% of train data)
y_pred_RandomForest = Random_Forest.predict(X_te)
scores_RandomForest = metrics.accuracy_score(y_te, y_pred_RandomForest)
scores_RandomForest
```

```
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:   13.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:   30.1s finished
```

```
0.9112993974070549
```

- Although the Overall Accuracy is high but due to class imbalances accuracy is not the best metrics to quantify the classifier’s performance.

- Precision, Recall and F1 Score

```
from sklearn.metrics import classification_report
print(classification_report(y_te,y_pred_RandomForest))
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95   1065128
           1       0.87      0.16      0.26    121245

   micro avg       0.91      0.91      0.91   1186373
   macro avg       0.89      0.58      0.61   1186373
weighted avg       0.91      0.91      0.88   1186373
```

- F1- Score Class 1 =0.26
- Precision Class1=87%
- Recall Class1 = 16%

Confusion Matrix

```
pd.crosstab(y_te,y_pred_RandomForest, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	All
True			
0	1062280	2848	1065128
1	102384	18861	121245
All	1164664	21709	1186373

- **Out of 1065128 instances of a product not being ordered -**
  - 1062280 times the classifier was correctly able to predict that the product would not be reordered
  - 2848 times the classifier misclassified a not ordered product as reordered product.
- **Out of 121245 instances of a product being reordered -**
  - 102384 times the classifier misclassified a reordered product as not currently ordered.
  - 18861 times the classifier correctly classified product as reordered

Random Forest Model Metrics on Test Data

- Model Accuracy = 90.31%
- Precision, Recall , F1 of Random Forest on Test Data

```
from sklearn.metrics import classification_report
print(classification_report(y_te_1,y_pred_test_RF))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95   1786248
           1       0.61      0.12      0.20    200771

   micro avg       0.90      0.90      0.90   1987019
   macro avg       0.76      0.56      0.57   1987019
weighted avg       0.88      0.90      0.87   1987019
```

On the test dataset:

- Class 0 : Precision = 91% | Recall =99% | F1 Score=0.95
- Class 1 : Precision = 61% | Recall =12% | F1 Score=0.20

- Confusion Matrix

```
pd.crosstab(y_te_1,y_pred_test_RF, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	All
True			
0	1770742	15506	1786248
1	176928	23843	200771
All	1947670	39349	1987019

6.Feature Selection

- Feature Selection is the process of selecting out the most significant features from a given dataset.
- Importance of feature selection :
  - It enables the machine learning algorithm to train faster.
  - It reduces the complexity of a model and makes it easier to interpret.
  - It improves the accuracy of a model if the right subset is chosen.
- In the current dataset – 2 types of feature selection are done
  - Recurrent Feature Elimination
  - Feature Importance

Feature Importance Method

- Feature importance of each feature of the dataset can be obtained by using the feature importance property of the model.
- It gives score for each feature of the data, the higher the score more important is the feature towards output variable.
- Feature importance is an inbuilt class that comes with Tree Based Classifiers. Here Random Forest is being used.

```
print(RandomForest.feature_importances_)
```

```
[0.04797194 0.03634959 0.04564338 0.03480491 0.02720089 0.03377166
 0.0350788  0.04009387 0.03391157 0.03384475 0.03412726 0.03139867
 0.03220216 0.04566014 0.04786582 0.04707142 0.04650463 0.09905013
 0.03801115 0.0454801  0.04341607 0.04434399 0.04585245 0.00022412
 0.00071467 0.00200777 0.00234493 0.00146426 0.00019144 0.00141197
 0.00339857 0.00191169 0.00123798 0.00252655 0.00113706 0.00056259
 0.00142082 0.00058744 0.0001823  0.00188839 0.00065056 0.00040345
 0.00339851 0.00267961]
```

- Most important features (Top 11)

features	imp score
'user_product_order_freq',	0.09905013
'user_avg_days_since_prior_order',	0.04786582
'user_avg_order_dow',	0.04707142
'user_avg_order_hour_of_day',	0.04650463
'product_avg_days_since_prior_order_per_user',	0.04585245
'user_total_products',	0.04566014
'user_product_total_orders',	0.04564338
'product_avg_add_to_cart_order_delta_per_user',	0.0454801
'product_avg_order_hour_of_day_per_user',	0.04434399
'product_avg_order_dow_per_user',	0.04341607
'product_avg_add_to_cart_order',	0.04009387

- Subset training data to only contain top 11 most important features. (**X_tr_fi**)

- Train logistic regression on the new subsetted dataset (X_tr_fi)

```
#logistic regression with 11 imp features
lr = LogisticRegression(class_weight='balanced')
log_12_imp_fe=lr.fit(X_tr_fi, y_tr)

#Predicting for test data called X_te (this is obtained by splitting 20% of train data)
y_pred_LR_12_fi = log_12_imp_fe.predict(X_te_fi)
scores_LR_12_fi = metrics.accuracy_score(y_te, y_pred_LR_12_fi)
scores_LR_12_fi
```

```
0.8556861964997518
```

- Model Accuracy of feature Importance model is 85.56%
- **Performance Metrics**
- F1- Score Class 1 =0.39
- Precision Class1=34%
- Recall Class1 = 45%

```
from sklearn.metrics import classification_report
print(classification_report(y_te,y_pred_LR_12_fi))
```

	precision	recall	f1-score	support
0	0.94	0.90	0.92	1065128
1	0.34	0.45	0.39	121245
micro avg	0.86	0.86	0.86	1186373
macro avg	0.64	0.68	0.65	1186373
weighted avg	0.87	0.86	0.86	1186373

Confusion Matrix

```
pd.crosstab(y_te,y_pred_LR_12_fi, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	All
True			
0	960531	104597	1065128
1	66613	54632	121245
All	1027144	159229	1186373

- **Out of 1065128 instances of a product not being ordered -**
  - 960531 times the classifier was correctly able to predict that the product would not be reordered
  - 104597 times the classifier misclassified a not ordered product as reordered product.
- **Out of 121245 instances of a product being reordered -**
  - 66613 times the classifier misclassified a reordered product as not currently ordered.
  - 54632 times the classifier correctly classified product as reordered

Recursive Feature Elimination Method (or RFE)

- RFE works by recursively removing attributes and building a model on those attributes that remain.
- It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

```
from sklearn.feature_selection import RFE
lr = LogisticRegression(class_weight='balanced')
rfe = RFE(lr, 5)
fit = rfe.fit(X_tr, y_tr)
print("Num Features: %d") % fit.n_features_
print("Selected Features: %s") % fit.support_
print("Feature Ranking: %s") % fit.ranking_
```

```
Num Features: 5
Selected Features: [False False  True False False False False  True  True False False False
 False  True False False False  True False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False]
Feature Ranking: [22 25  1 13 20 18  4  1  1 15  3  6  2  1 12 39 23  1 10 24 19 17 40 11
 30 29 32 36 34 38 27 28 37 33 16  9 35  5 14  8  7 21 26 31]
```

- Top 5 important features according to RFE Method

Feature	value
'user_product_total_orders',	True
'product_avg_add_to_cart_order',	True
'product_avg_order_dow',	True
'user_total_products',	True
'user_product_order_freq',	True

- Model Accuracy

```
y_pred_LR_RFE = fit.predict(X_te)
scores_LR_RFE = metrics.accuracy_score(y_te, y_pred_LR_RFE)
scores_LR_RFE
```

```
0.8654504106212801
```

  - Model Accuracy of logistic Regression with top 5 features (RFE method) is 86.5%

- **Precision , Recall , F1 Score**
  - F1- Score Class 1 =0.39
  - Precision Class1=36%
  - Recall Class1 = 42%

```
from sklearn.metrics import classification_report
print(classification_report(y_te,y_pred_LR_RFE))
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92   1065128
           1       0.36      0.42      0.39    121245

   micro avg       0.87      0.87      0.87   1186373
   macro avg       0.65      0.67      0.66   1186373
weighted avg       0.87      0.87      0.87   1186373
```

- **Confusion Matrix:**

```
pd.crosstab(y_te,y_pred_LR_RFE, rownames=['True'], colnames=['Predicted'], margins=True)
```

Predicted	0	1	All
True			
0	976033	89095	1065128
1	70531	50714	121245
All	1046564	139809	1186373

- **Out of 1065128 instances of a product not being ordered -**
  - 976033 times the classifier was correctly able to predict that the product would not be reordered
  - 89095 times the classifier misclassified a not ordered product as reordered product.
- **Out of 121245 instances of a product being reordered -**
  - 70531 times the classifier misclassified a reordered product as not currently ordered.
  - 50714 times the classifier correctly classified product as reordered

Performance Comparison

Model Type	Model Accuracy	Precision	Recall	F1 Score
	Overall Accuracy of the Model in correctly classifying	Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all products that labeled as reordered by the model, how many were reordered in reality	Recall is the ratio of correctly predicted positive observations to the all observations in actual class. The question recall answers is: Of all the products that truly reordered, how many did the model label as reorder	F1 Score is the weighted average of Precision and Recall.
Logistic Regression	85.22%	**Class 0:** 94% **Class 1:** 34% **Weighted :** 88%	**Class 0:** 90% **Class 1:** 47% **Weighted** : 85%	**Class 0:** 0.92 **Class 1:** 0.39 **Weighted** : 0.86
Random Forest	91.12%	**Class 0:** 91% **Class 1:** 87% **Weighted** : 91%	**Class 0:** 100% **Class 1:** 16% **Weighted** : 91%	**Class 0:** 0.95 **Class 1:** 0.26 **Weighted** : 0.88
Logistic Regression (feature Importance)	85.56%	**Class 0:** 94% **Class 1:** 34% **Weighted :** 87%	**Class 0:** 90% **Class 1:** 45% **Weighted** : 86%	**Class 0:** 0.92 **Class 1:** 0.39 **Weighted** : 0.86
Logistic Regression(RFE)	86.54%	**Class 0:** 93% **Class 1:** 36% **Weighted** :87%	**Class 0:** 92% **Class 1:** 42% **Weighted** :87%	**Class 0:** 0.92 **Class 1:** 0.39 **Weighted** :0.87

7.Predicting Test Dataset Results

order_id	product_id	user_id	Prediction
393	6184	111860	1
393	13424	111860	0
393	12078	111860	1
393	16797	111860	1
393	19828	111860	1
393	30591	111860	1
393	32403	111860	1
473	47144	77529	1
473	20082	77529	1
473	36441	77529	0
631	42265	184099	1
631	21137	184099	1
631	27344	184099	1
631	13829	184099	0
631	15842	184099	0
631	9203	184099	0
774	43335	27650	0
774	16108	27650	0
1280	49235	176046	1
1280	27845	176046	1
1280	39581	176046	1
1280	48186	176046	1

8.Steps to improve Model Performance

- The training data has class imbalance. There are more instances for class 0(“ *product not being in latest order*”) than class 1 (“*product being in latest order”)*

```
y_tr.value_counts()
```

```
0     4265760
1      482402
```

- *Currently class_weight*='balanced‘ is used in training the logistic Regression. Which effectively tells that each class is equally important.
- However, Oversampling and under sampling will create equal number of instances for both the classes
- *Conduct* **SMOTE (Synthetic Minority Over-sampling Technique)**
  - By creating synthetic (not duplicate) samples of the minority class. Thus making the minority class equal to the majority class.
- *Conduct NearMiss*
  - This is an under-sampling technique. Instead of resampling the Minority class, this will make the majority class equal to minority class.

9.Insights & Recommendations

- Improved Stocking and restocking of products
- Coupons for boosting midweek sales
  - Increasing Sales of Popular Departments in mid week through targeted coupons. Thus increasing total sales
- Increasing product based loyalty of popular products

- Entire Solution is available at

https://github.com/SanchariChowdhuri/Instacart_Data_Analysis

Thank you