# 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 |

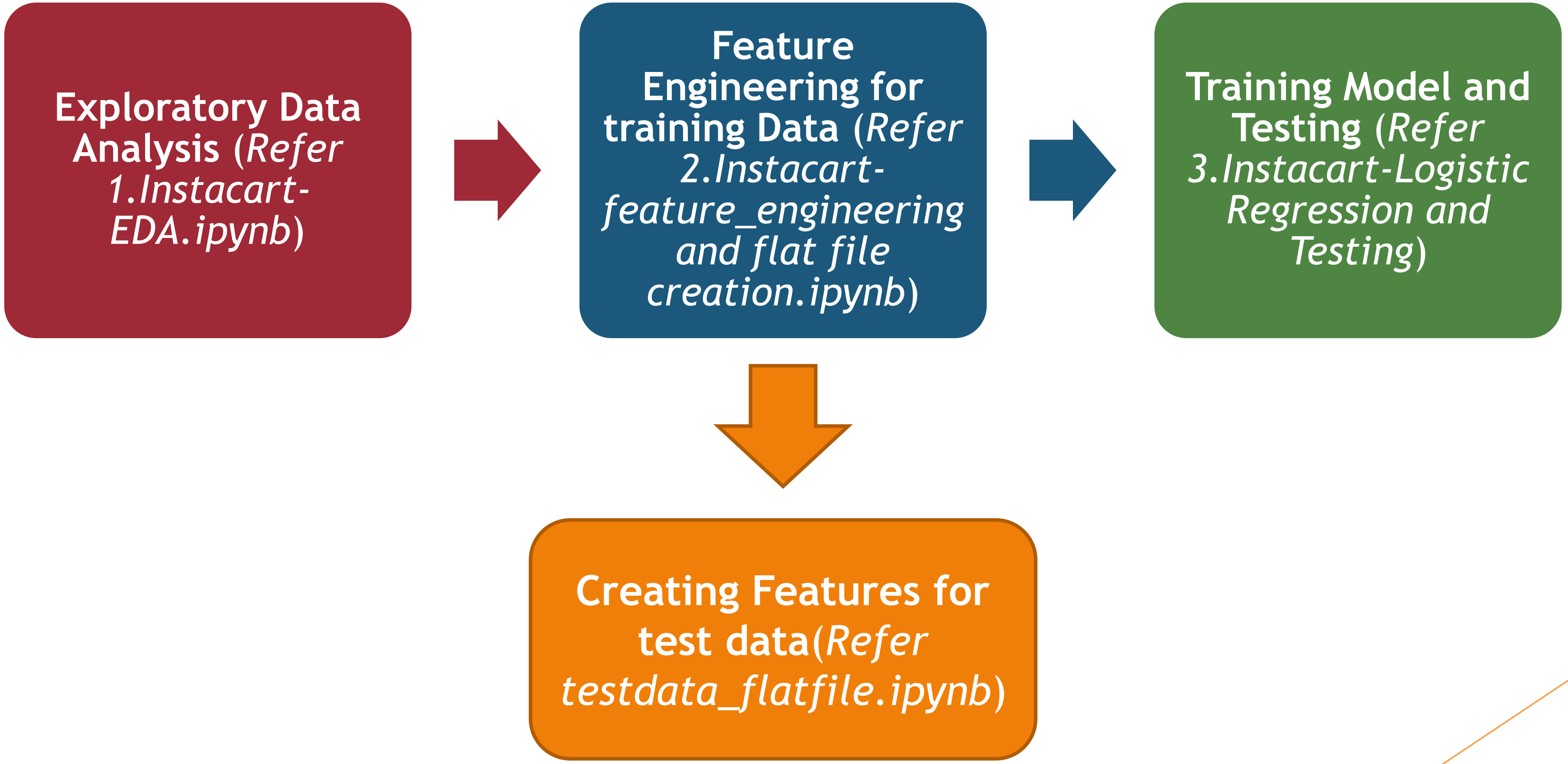
```
aisles_df.head()
```

| | 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()
```

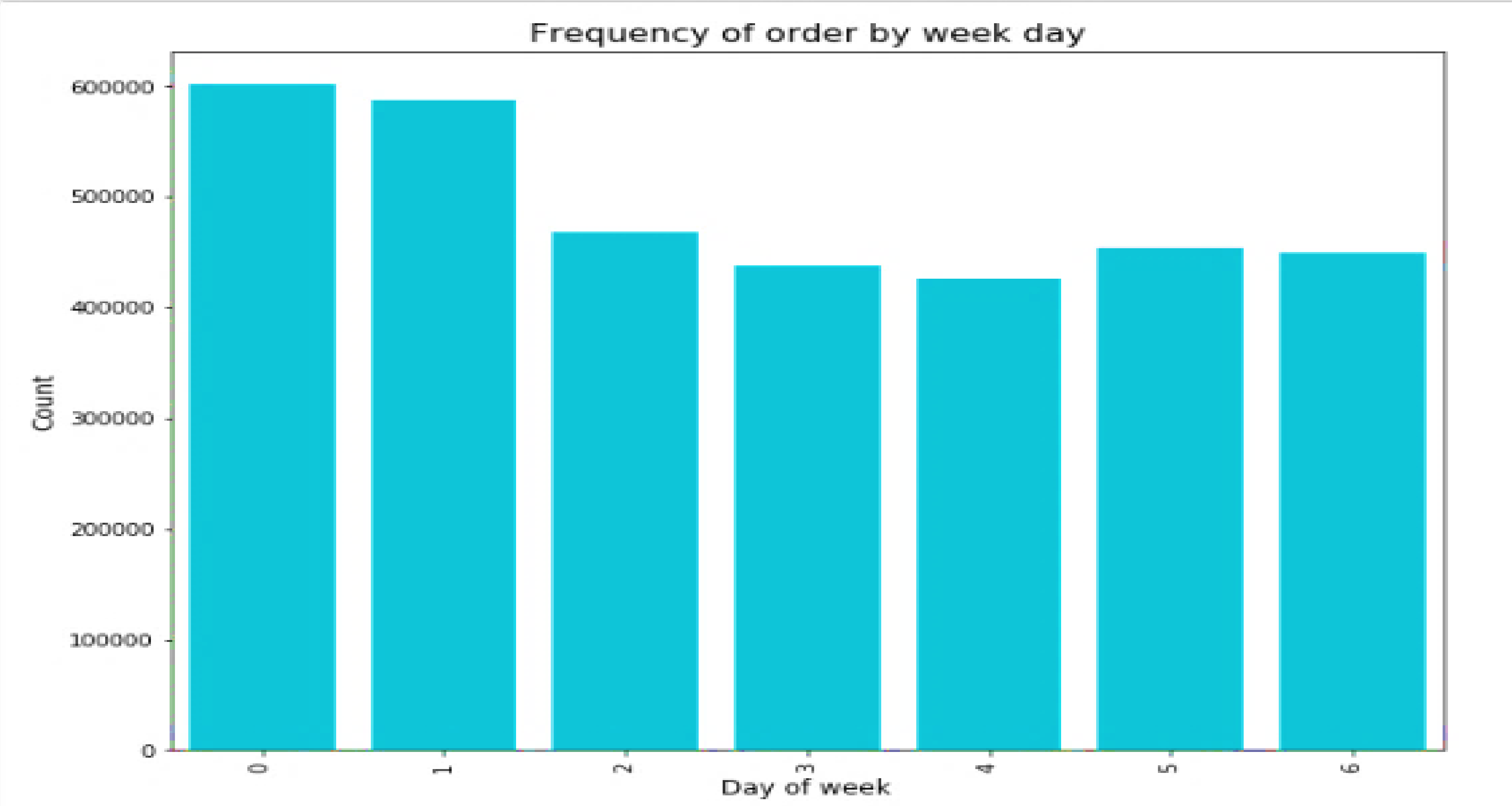
| | department_id | department |
|---|---|---|
| 0 | 1 | frozen |
| 1 | 2 | other |
| 2 | 3 | bakery |
| 3 | 4 | produce |
| 4 | 5 | alcohol |

# 2.Analysis Flow



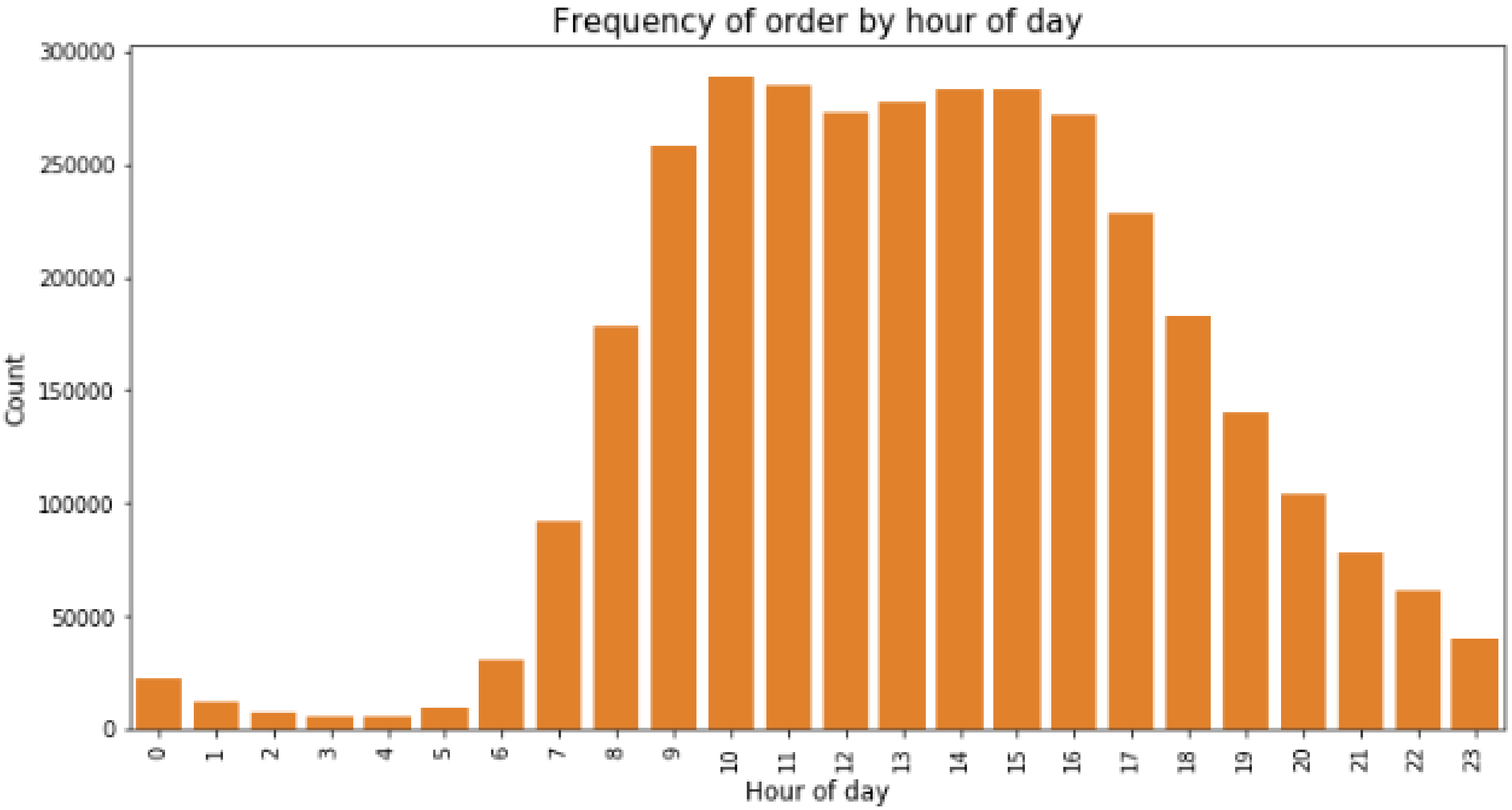
# 3. Exploratory Data Analysis

# Number of Orders per day of week



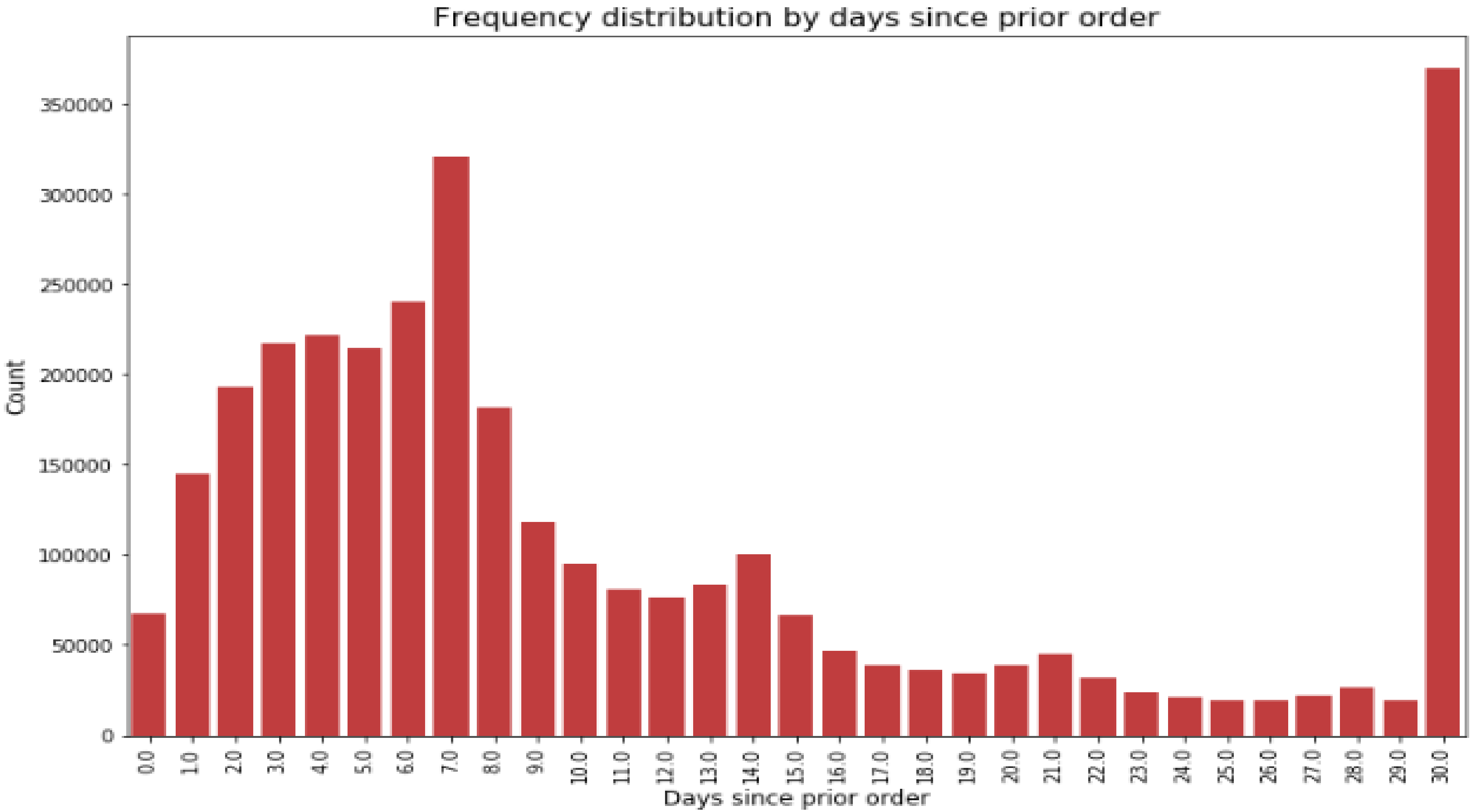

- *Sunday and Monday seems to have higher volume of orders than other days of the week*
- *Towards mid week there  is a dip in order volume suggesting people prefer to get their groceries towards weekend or beginning of the week*

# Orders by hours of the Day



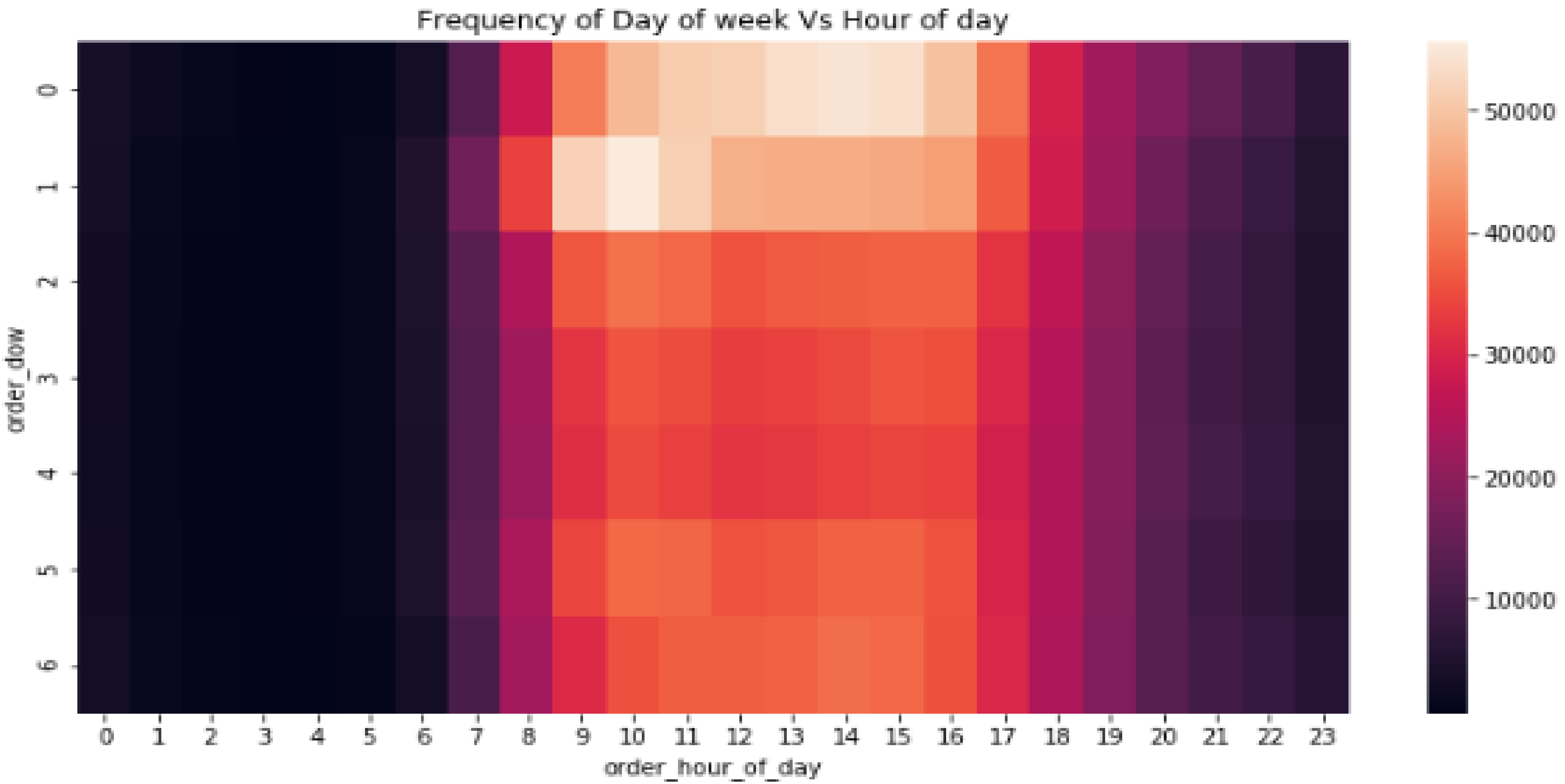

- *People order groceries in day time*

# How frequently do customers order?



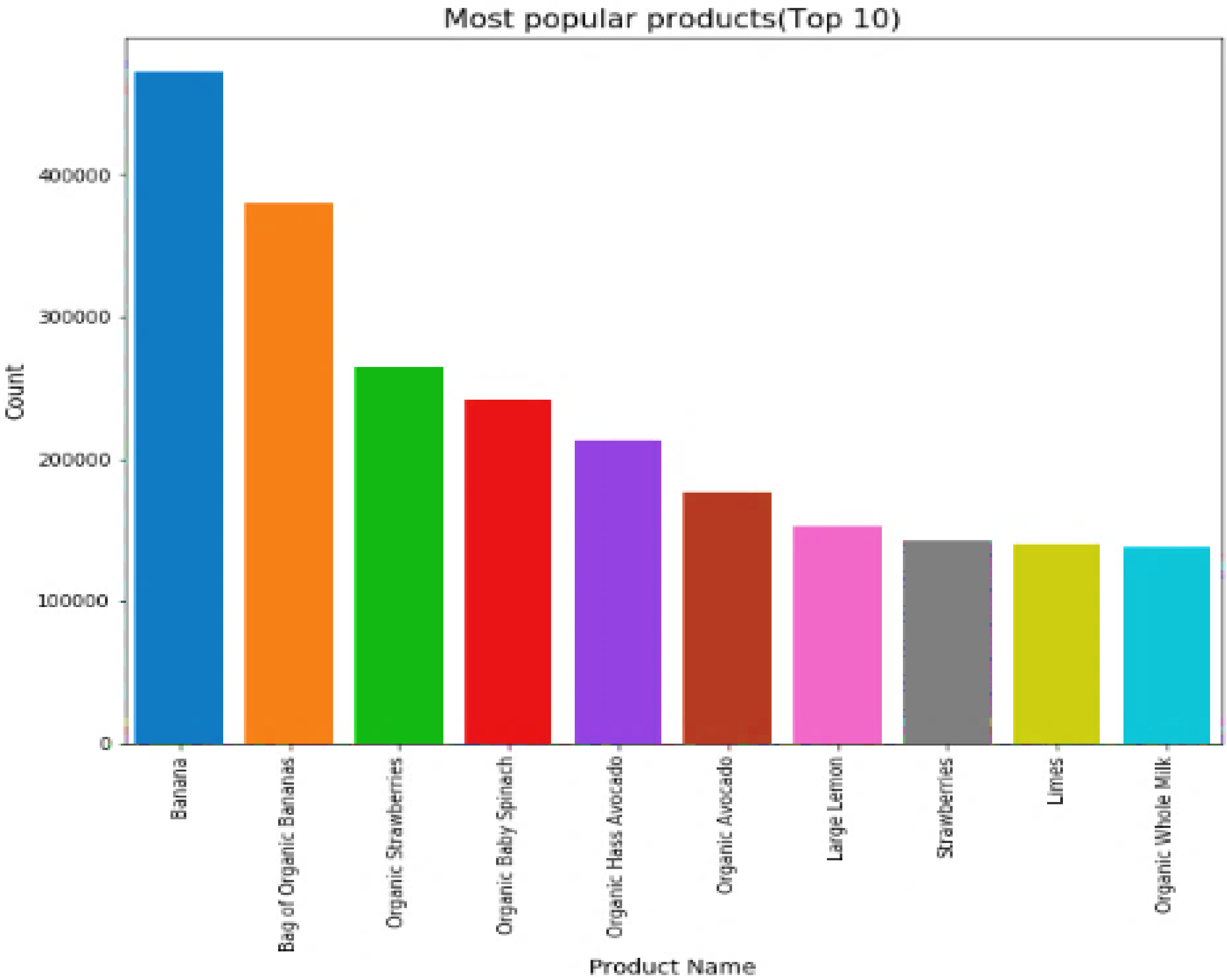

*Customers order Weekly and monthly the most*

# When do customers order the most

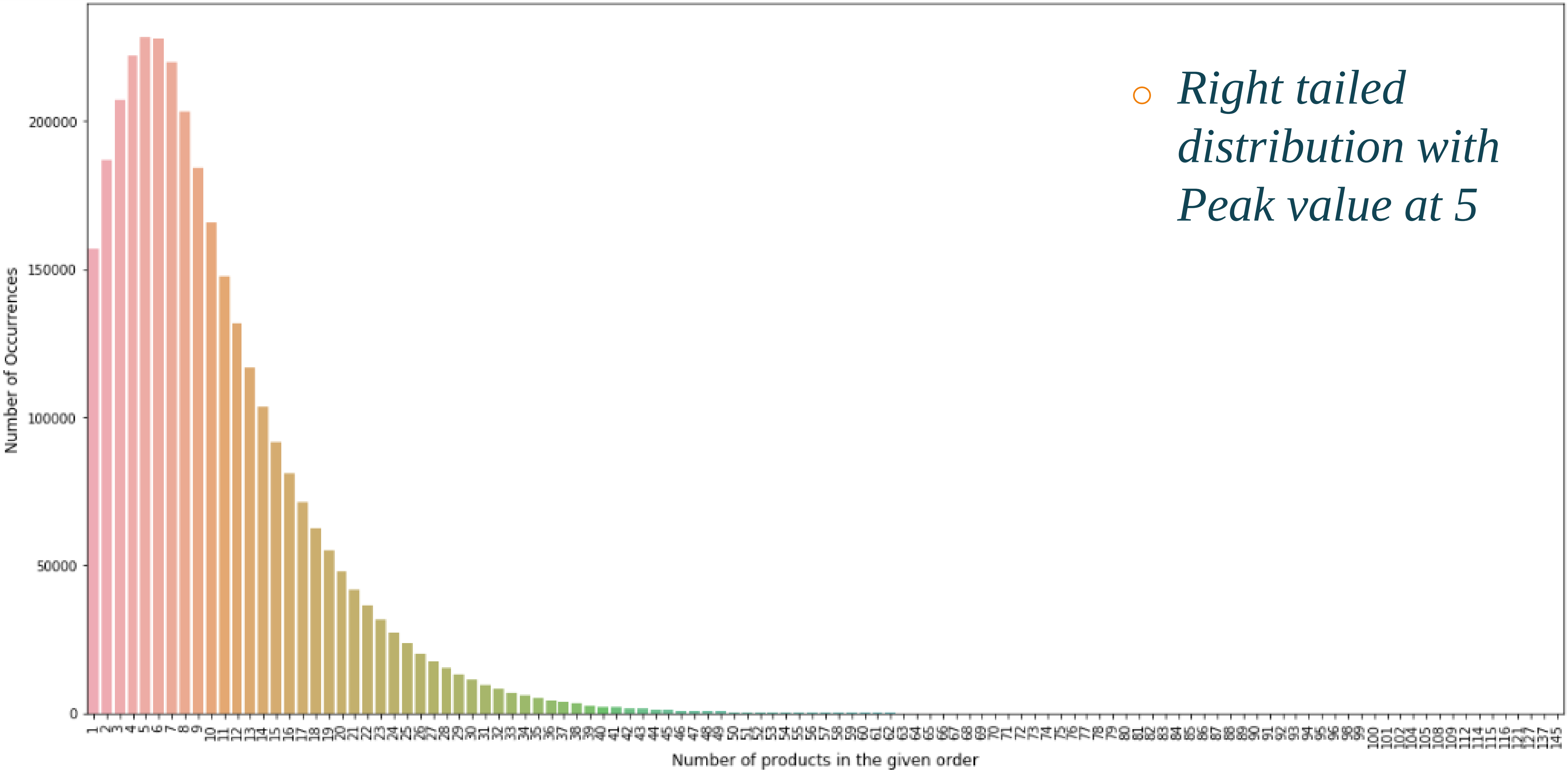



- *Heatmap showing volume of orders across hour of day and day of week*

# Most Popular products

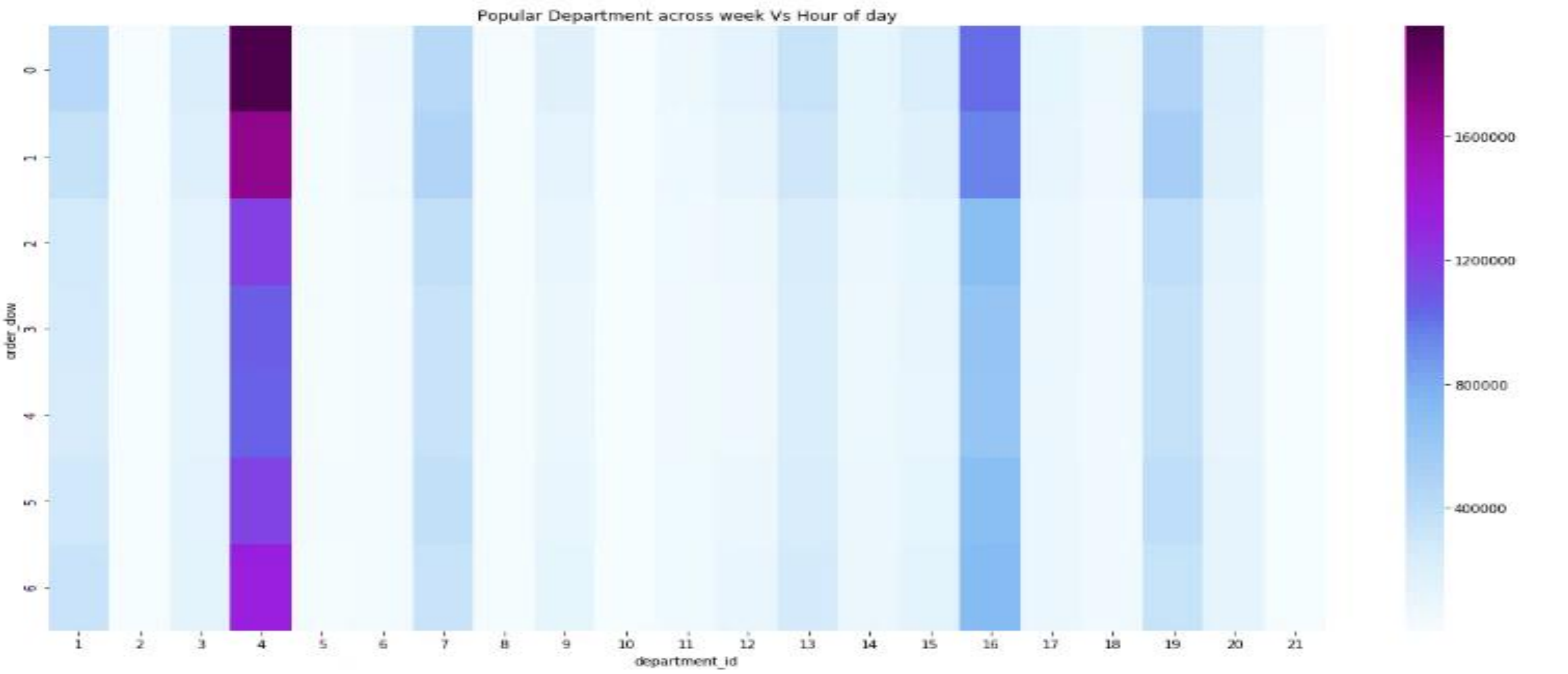


# Number of Products in an Order




- *Right tailed distribution with Peak value at 5*

# Popular Department across day of week and hours




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

# Departmentwise reorder ratio


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

# 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* |

# Training Model

- Model Used :Logistic Regression
- Independent variable = in_cart [0,1]
- Dependent 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 / True | 0 | 1 | All |
|---|---|---|---|
| 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 |

# Next Steps….

- 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.

Thank you