# **PROJECT REPORT**

# **Topic**: Instacart Customer Order Analysis

# **Author:** Harshit Mittal

# 

**Appendix**

Introduction……………………………………………………………………………………3

Description of Dataset………………………………………………….……………………...4

Data Source…………………………………………………………….……………………...5

Methodology…………………………………………………………...……………………...5

Data cleaning and Pre-processing……………………………………...……………………...6

Exploratory Data Analysis……………………………………………..……………………...7

Feature Engineering………………………………………………………………………….14

Modelling strategies…………………………………………………………………………17

Result and analysis…………………………………………………………………………..21

Conclusion……………………………………………………………..……………………25

Future Work………………………………………………………………………………....25

References……………………………………………………………..…………………….25

**Instacart Customer Order Analysis**

**Introduction:-**

With the advancement in technology, online retail shopping has become a necessity and more and more people and businesses are joining the online market. This also gives organizations an opportunity to better understand their customer and tailor their operations according to customers’ needs for better satisfaction rating and to retain customers. Data science team, in any organization, plays a big part in providing a delightful shopping experience to its customers. Analyzing transaction details of users is a key approach to better understand customer purchasing pattern. Instacart is an online grocery ordering and delivery service, based in US & Canada with over 500 million products and 40,000 stores. Ordering food supplies online is a new way of restocking groceries and other essential items. Ordering groceries online is a hassle-free exercise in today’s world. But no organization wants a situation where a customer forgets to order an item which they would normally order from their platform. To avoid such situations, online retailers provide the customers with product suggestions based on their past purchase history and patterns. They have open-sourced their transactional data which can be used for analysis and predictions.

Crucial benefits of such analysis are that it will lead to increase in sales for businesses, better identify products that can be sold together, improve inventory management. Moreover, it also enhances user’s experience by providing relevant suggestions to them. Data science has emerged as a lifesaver that can be used to make better informed business decisions, and to predict trends. This has served as a boon as the retail sector strives to stay technologically relevant while meeting customer demands. With help of data science, we can visualize customer behavior, and hence retailers can foresee customer likes/dislikes. This power of data analysis enables organizations to serve their customers much more efficiently. The insights also allow businesses to identify high-value customers, their motivations to buy particular products, and so on, which can not only enhance customer acquisitions but also boost customer loyalty.

The objective of our project is to predict which products a user may want to buy in their next order. Instacart open sourced their transactional data of over 3 million orders, from more than 200,000 Instacart users. We shall use this anonymized data on customer orders over time to predict which previously purchased products will be in a user’s next order. The predictions depend on historical data, leading up to the most recent transactions. Meanwhile, also analyzing the customer data to gain some useful insights about the purchasing pattern, which will be powerful in determining future business operations. With the integration of machine learning algorithms and data analysis platforms, we can impart retailers with robust predictive analytics capabilities, enabling them to stock their stores with the right products at the right time. The algorithms allow the retailers to detect patterns in the various operations and processes of the supply chain. Some additional marketing strategies that retailers might come up with, may include:

* Design better Product Catalog
* Cross marketing on online stores
* Roll-out customized emails with add-on sales, etc.

**Description of Dataset:-**

The data contains anonymized sample of over 3 million orders from more than 200,000 users. For each user, Instacart provided between 4 and 100 of their orders, along with the sequence in which products were placed in the cart. There are a total of 6 files with a total of ***207*** *Megabytes* of data. The entire dataset can be broadly categorized into the following:

1. Prior data: Order history of every user. This data contains nearly 3-100 past orders per user (~3.2m orders)
2. Train data: Current order data of every user. This data contains only 1 order per user (~131k orders)
3. Test data: Future order data of every user . This data will not contain any product information. Essentially, there are the orders for which we have to predict the reorder products for users. (75k orders)

We want to predict which previously purchased products (prior orders) will be in a user’s next order (train and test orders). For the train orders, Instacart has revealed the results (i.e., the ordered products) while for the test orders we do not have this piece of information. Moreover, the future order of each user can be either train or test meaning that each user will be either a train or a test user. This information of order type is specified in the ‘Orders.csv’ file, under the ‘eval\_set’ column. The setting of the Instacart problem is described in the figure below.

The files are as follows:-

1. aisles.csv - This file contains aisle information. Shape of file is (134, 2).
   1. aisle\_id - Unique id for each aisle.
   2. aisle - Name of aisles.
2. departments.csv - This file contains department information. Shape of file is (21,2).
   1. department\_id - Unique id for each department.
   2. department - Name of departments.
3. order\_products\_train.csv – This file consists of all product details for a train order. Shape of file is (1384617, 4).
   1. order\_id - Unique id for each order.
   2. product\_id - Unique id for each product.
   3. add\_to\_card\_order - specifies the sequence of added products.
   4. reordered - whether a product was reordered or not.
4. order\_products\_prior.csv - These files contain data about which products were purchased in each order. Shape of file is (32434489, 4)
   1. order\_id - Unique id for each order.
   2. product\_id - Unique id for each product.
   3. add\_to\_card\_order - specifies the sequence of added products.
   4. reordered - whether a product was reordered or not.
5. orders.csv - This file contains prior, train, test order details placed by any user. Shape of file is (3421083, 7)
   1. order\_id - Unique id for each order.
   2. user\_id - Unique id for each user.
   3. eval\_set - Category of order (prior/train/test)
   4. order\_number - order sequence of a user.
   5. order\_dow - day of the week when order placed.
   6. order\_hour\_of\_day - Time of the day when order placed.
   7. days\_since\_prior\_order - Number of days between two orders.
6. products.csv - This file contains product details. Shape of file is (49688, 4)
   1. product\_id - Unique id for each product.
   2. product\_name - Name of products.
   3. aisle\_id - Unique id of aisle where product is kept.
   4. department\_id - Unique id of department to which this product belongs.

**Data Source:-**

<https://www.kaggle.com/datasets/psparks/instacart-market-basket-analysis>

**Methodology:-**

We need to generate some rules and study patterns in order to give product recommendations with high probability. Explicitly coding all such rules and behaviors will get extremely cumbersome. We shall use Machine learning algorithms to achieve our goal. We shall automate the learning process to study the 3 million data points which Instacart has open sourced. Based on the order history and user preferences of a product, we will be able to predict the products which could be reordered.

Since, we have to predict multiple products for a given order, this might look like a multilabel classification task. There are 49688 products, and total product recommendations could be anywhere from None to N. Therefore, we will restructure this problem into a binary classification problem, where we will predict the probability of an item being reordered by a user. We shall classify each item into 2 categories for any particular user, i.e., if that product will be reordered or not.

To build a machine learning model, we need to extract additional features from orders data to understand user's purchase pattern and popularity of all the products. We shall extract following features from the user's transactional data.

* Product Level Features
* Aisle and Department Level Features
* User Level features
* User-product Level Features

Generally, we can say if Probability (item reorder) > 0.5 -> Class 1 (reorder in this case) else Class 0. So, we can select those products which belong to class 1 ( i.e., P(X) > 0.5) and recommend them to user. But this threshold 0.5 can be changed in order to improve the performance of the model.

We shall follow below processes to achieve our goal of predicting a reorder in an efficient manner:-

1. Data cleaning and pre-processing
2. Exploratory Data Analysis
3. Feature engineering
4. ML algorithms

**Data cleaning and Pre-processing:-**

Now that we have the dataset which provides numerous details regarding order journey of customers, we shall begin with cleaning the data and processing the same to make it eligible for solving the problem at hand. We shall scan and verify the data for below anomalies:-

* Look for duplicate values and perform deduplication.
* Find and fix any missing values.

1. Deduplication:

We shall load all the provided files of the dataset and look for duplicate values using the ‘duplicated()’ method. Post analyzing the files we found that no file has duplicate values. Hence, we are good to proceed for further verification on these files.

1. Missing values:

We will check if any file has NULL values in any of their column. It is pivotal to remove any null values from the dataset as such values hamper the model learning and degrade model prediction performance. To do so, we shall first analyze the null values in dataset (if any) and then fix them.

We can see none of the files have any null values, except ‘orders.csv’ file. The column ‘days\_since\_prior\_order’ has null values. Upon analyzing this data, we find out that there are total 206,209 null values in this column. This is the exact number of unique users in our dataset. Reason behind these null values is that for the first order of every user, value for days since prior order will be N/A. We shall replace these null values with 0 while preparing the dataset.

Now, that our data is clean, we can proceed to pre-process our data to formulate a working-set which helps to train the Machine leaning models efficiently. We also need to perform some data analysis to gather valuable insights from this data. So, we shall merge different files and select useful fields as per the requirement.

1. We shall merge ‘order\_products\_prior’ and ‘order\_products\_train’ dataset, to get a cumulative dataset comprising of all ‘train’ and ‘prior’ order-product details.
2. We shall now merge ‘orders’ and ‘order\_products\_merged’ dataset to club all the above order information with the complete order details.
3. We will also merge the above dataset with ‘products’, ‘aisles’, ‘departments’, and ‘orders’ datasets to get an all-encompassed dataset which contains all the order, product details for all orders (except test orders).
4. We shall merge ‘products’, ‘departments’, and ‘aisles’ dataset to get a comprehensive view of all details regarding products. We can now view the product names, respective departments which they belong to, respective aisles which they belong to, etc.

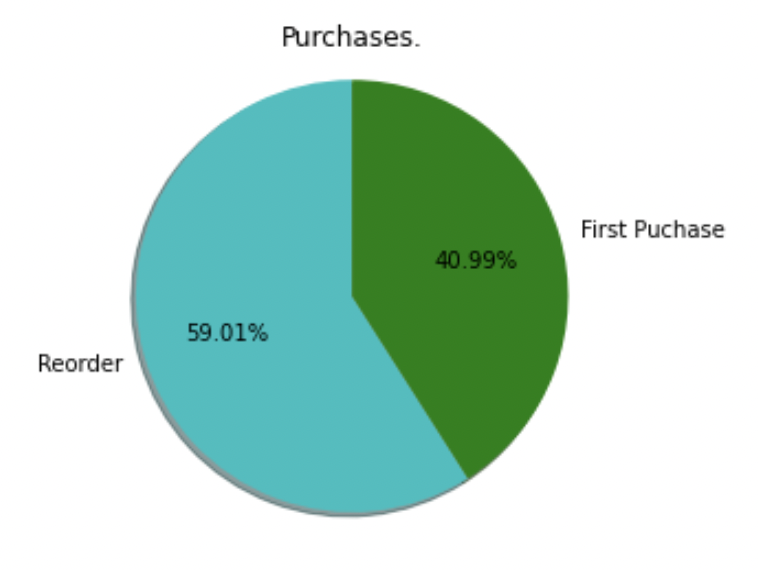
**Exploratory Data Analysis:**

This is an important first step in our analysis as it helps us uncover some powerful insights from the data and answer some interesting questions and can help us perform better feature engineering. It will also help us better understand user purchase patterns, which can be used by the organization to improve its operations. Our analysis helped us answer the following questions.

**Q1. How many percent of purchases are first purchase and how many are reorders.**

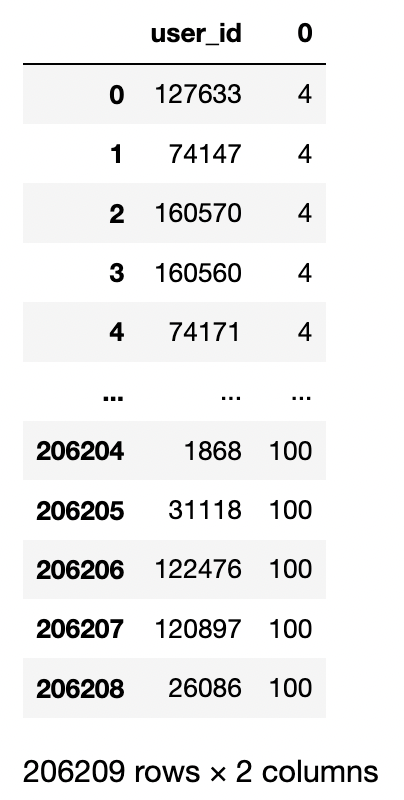


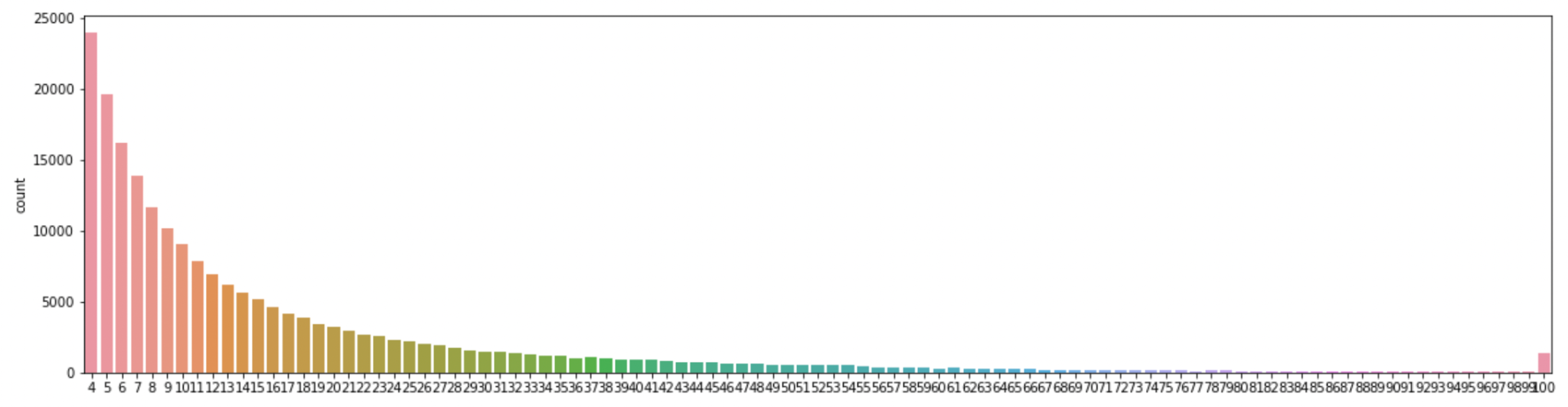




* From the pie chart it is evident that more than half, i.e., around 60% of the orders in the dataset are reorders.
* Only around 41% of the orders were first orders.

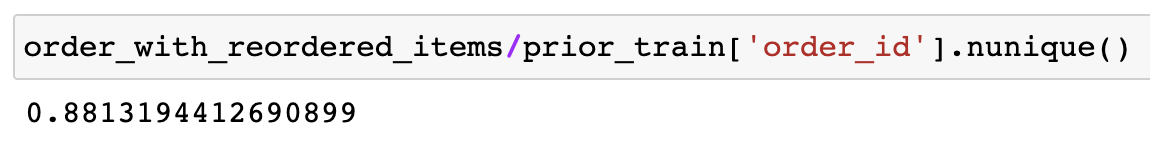
**Q2. Number of orders placed by each user.** 

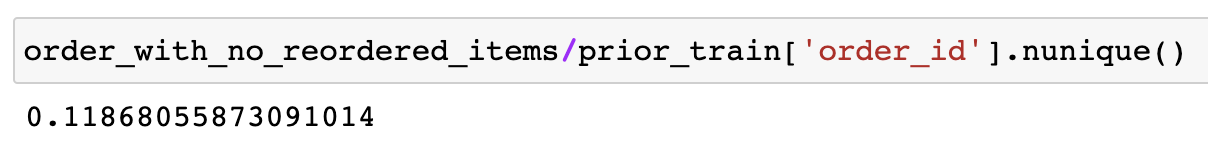




* Each user in the dataset have at least 4 orders and at most a 100.
* From the histogram it can be inferred that most users have placed order in the range 4-60.

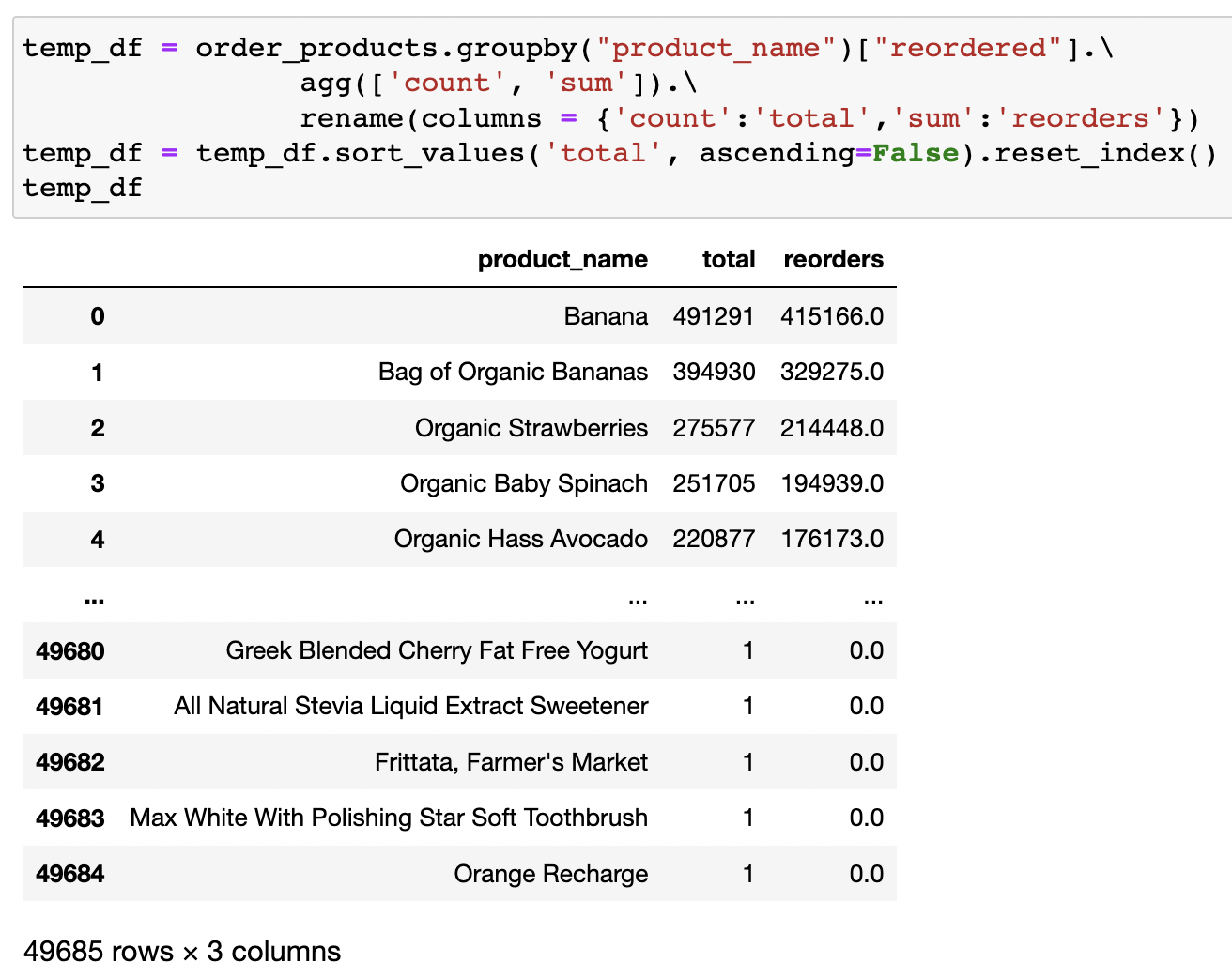
**Q3. How many Orders with no reordered products.**

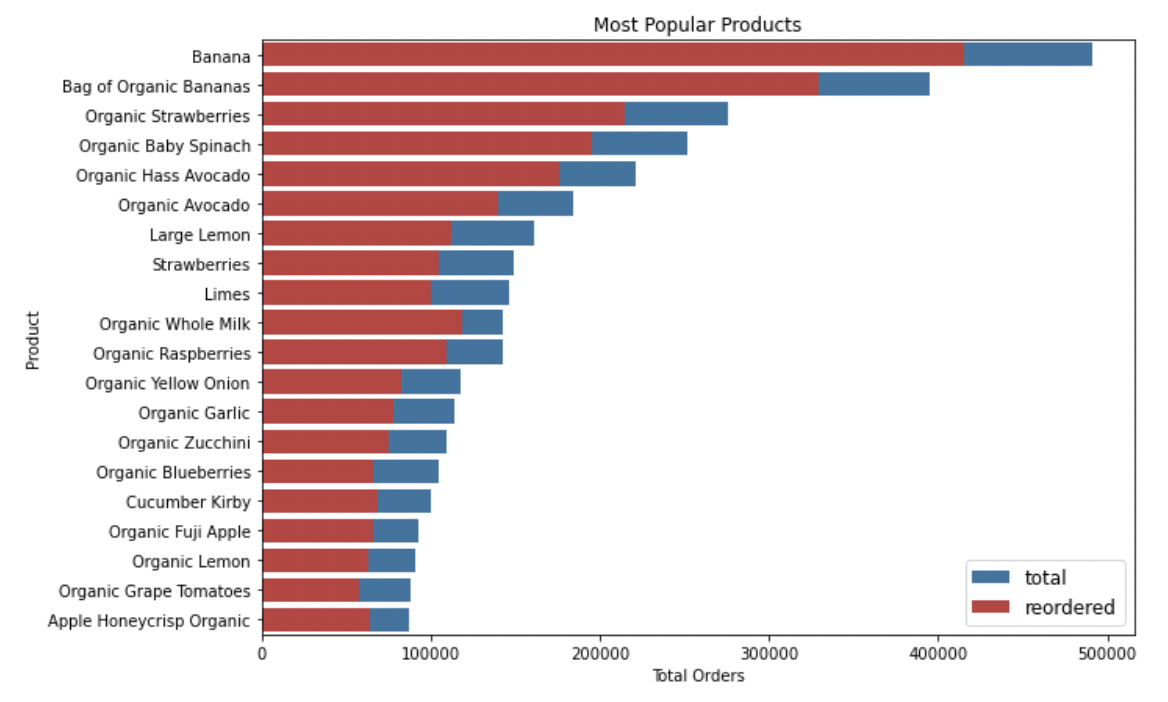




* Around 88% of orders have reordered items.
* Around 12% of orders have no reordered items.

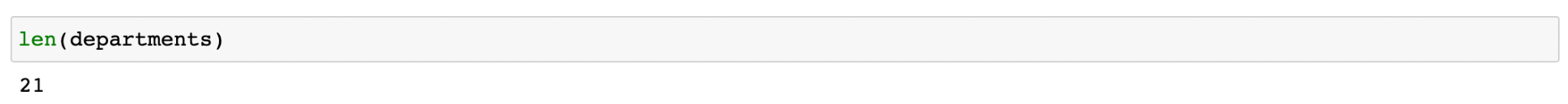
**Q4. Which are the most frequently ordered / reordered products.**

****

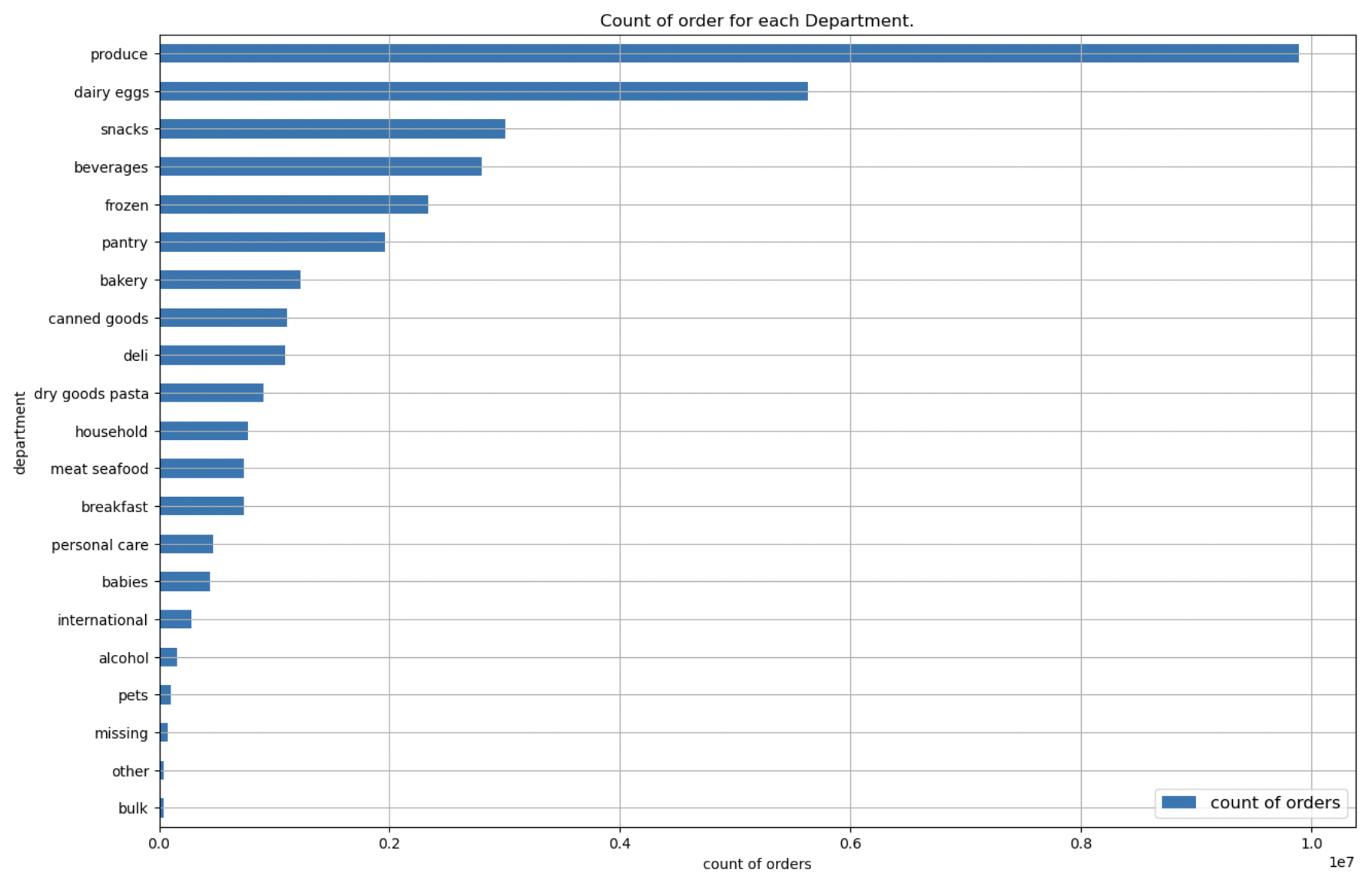
****

* A total of 49685 products were ordered.
* The most popular products (Top 5) are also organic in nature.
* Bananas are the most ordered and reordered product and most of the products which were ordered are organic products.

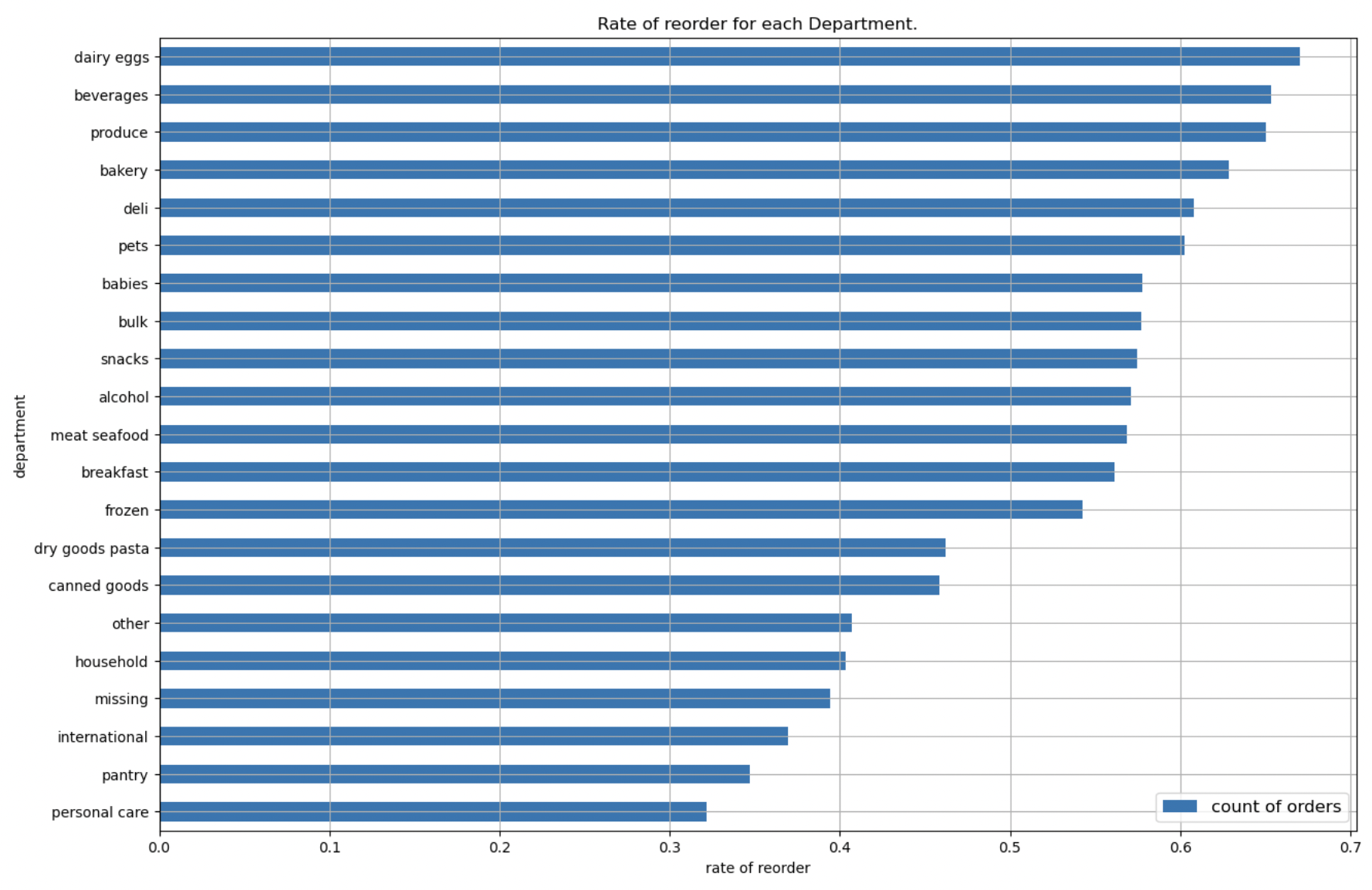
**Q5. Frequently ordered and reordered products categorized by department.**

****

* There are a total of 21 departments.



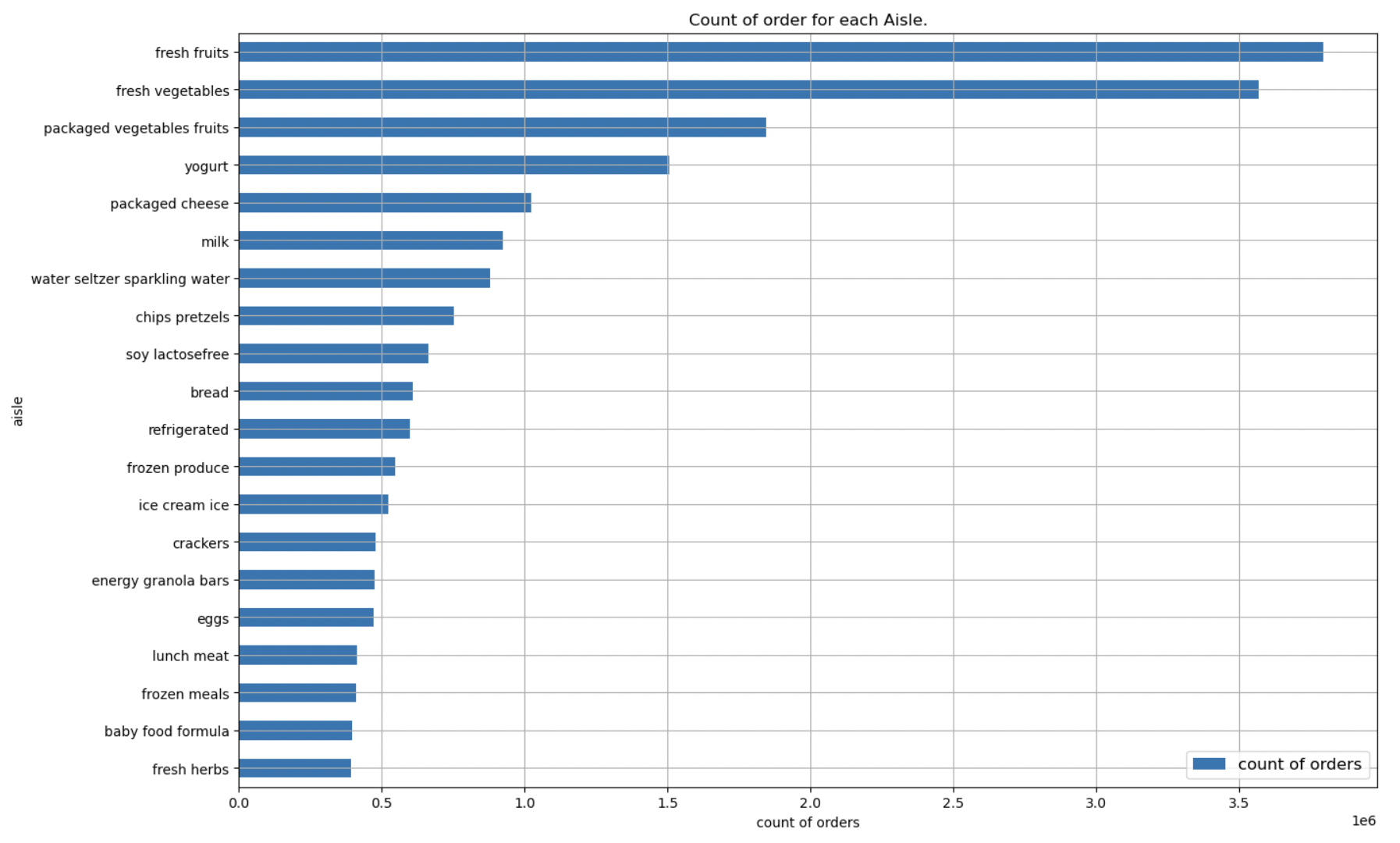
* Most of the ordered products are from the produce department which includes fruits, vegetables, etc.
* The least ordered items belong to the bulk department.



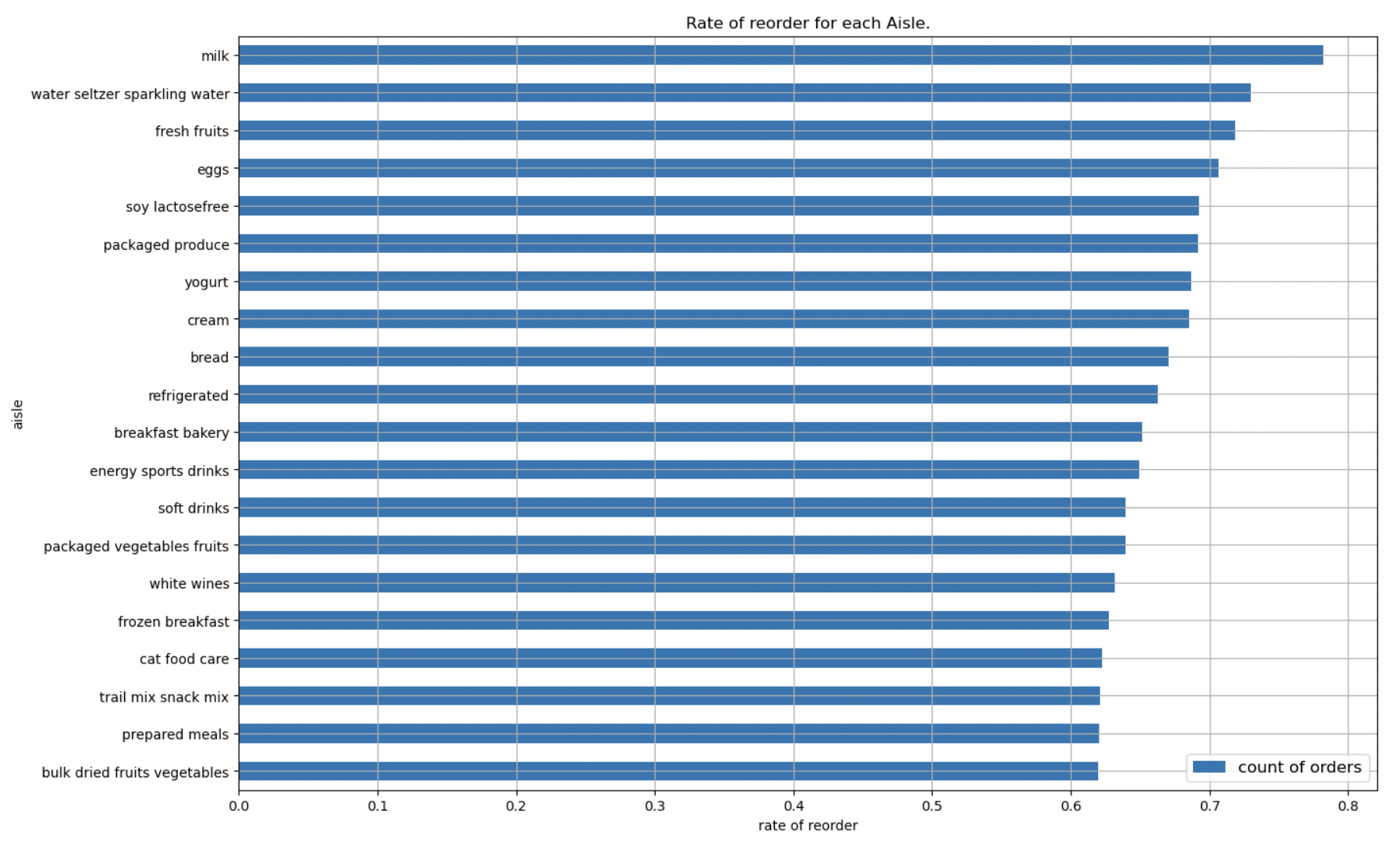
* There is high reorder rate in organic foods and daily consumables. Most reordered products belong to dairy eggs department which includes eggs, yogurt, cheese etc.
* There is low reorder rate in personal care department which makes sense as these items last longer and are used less often.

**Q6. Frequently ordered and reordered products categorized by aisle.**

* There are a total of 131 aisles.

****

* It can be seen that most ordered items are from fresh fruits and fresh vegetable aisles. Other frequently ordered aisles are packaged vegetables fruits, yogurt, and packaged cheese aisles.
* Items bought least frequently are from fresh herbs, baby food formula, frozen meals, lunch meat aisle.

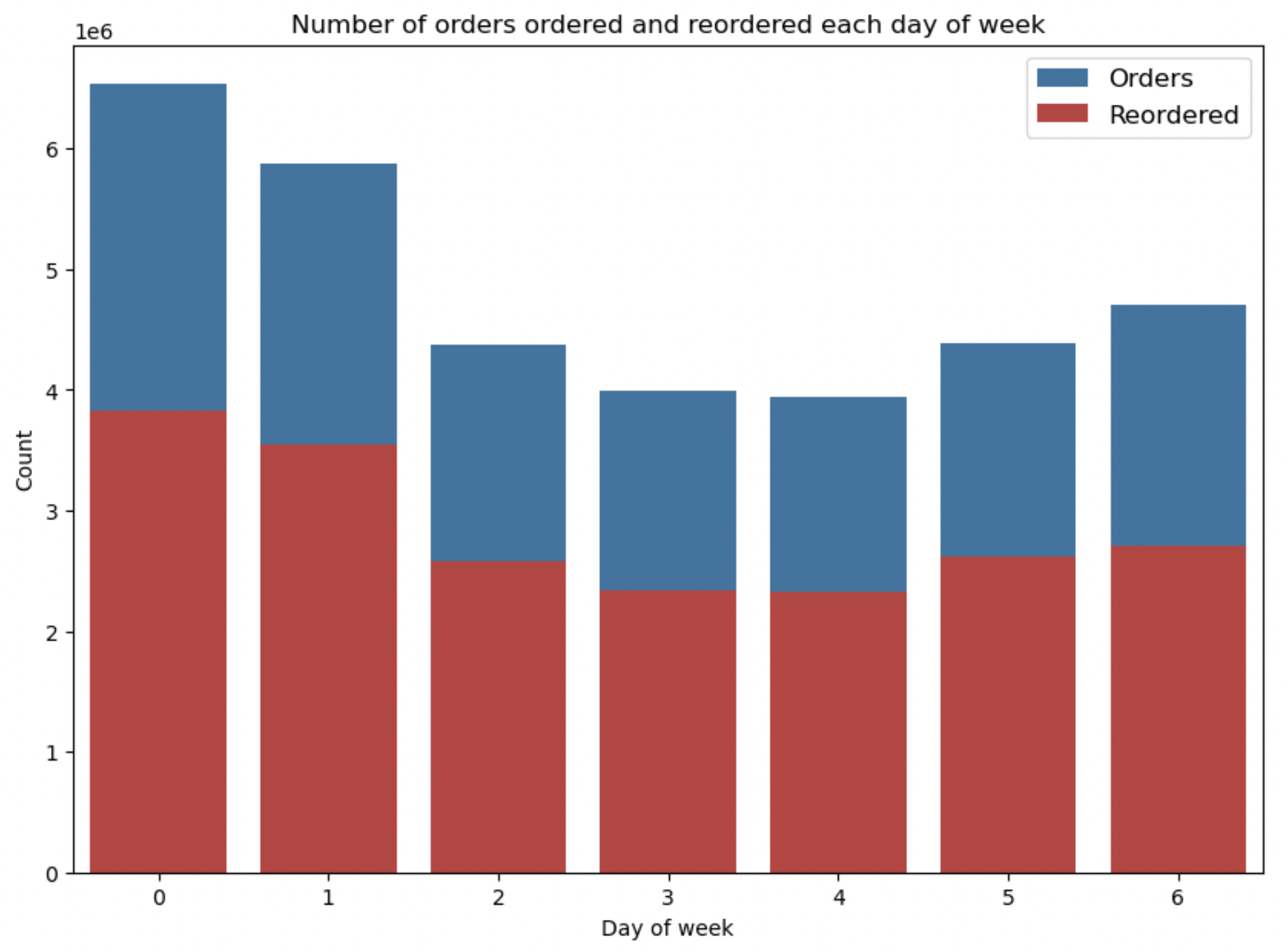


* Most reordered items are from aisles - milk, sparkling water, fruits, eggs. These items are daily used items which makes sense as one rarely ever changes their meal plan.
* Moreover, these are perishable products that don’t last very long.

**Q7. Number of products ordered and reordered in a particular day of the week.**

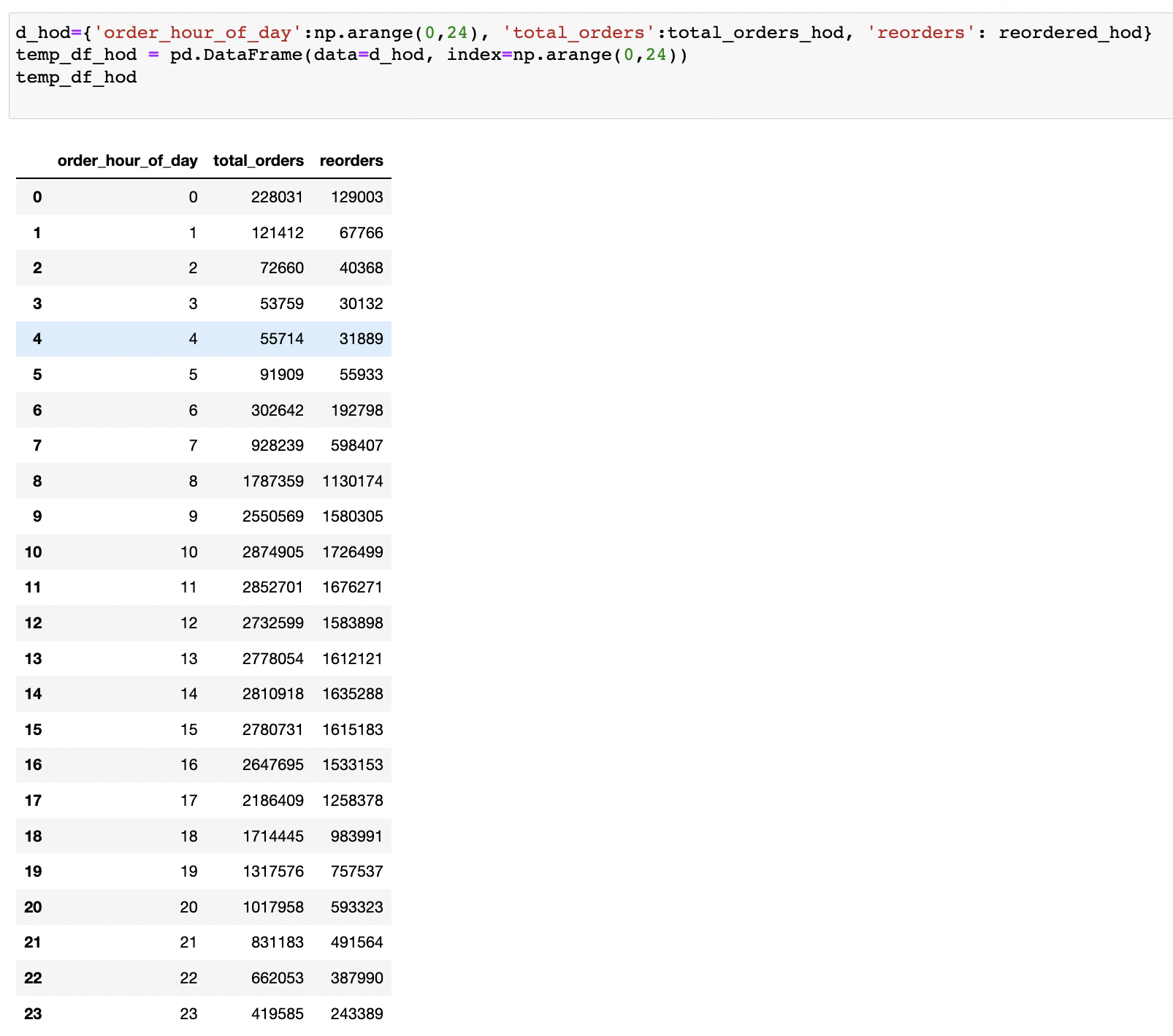
Graphical user interface, text, application, email

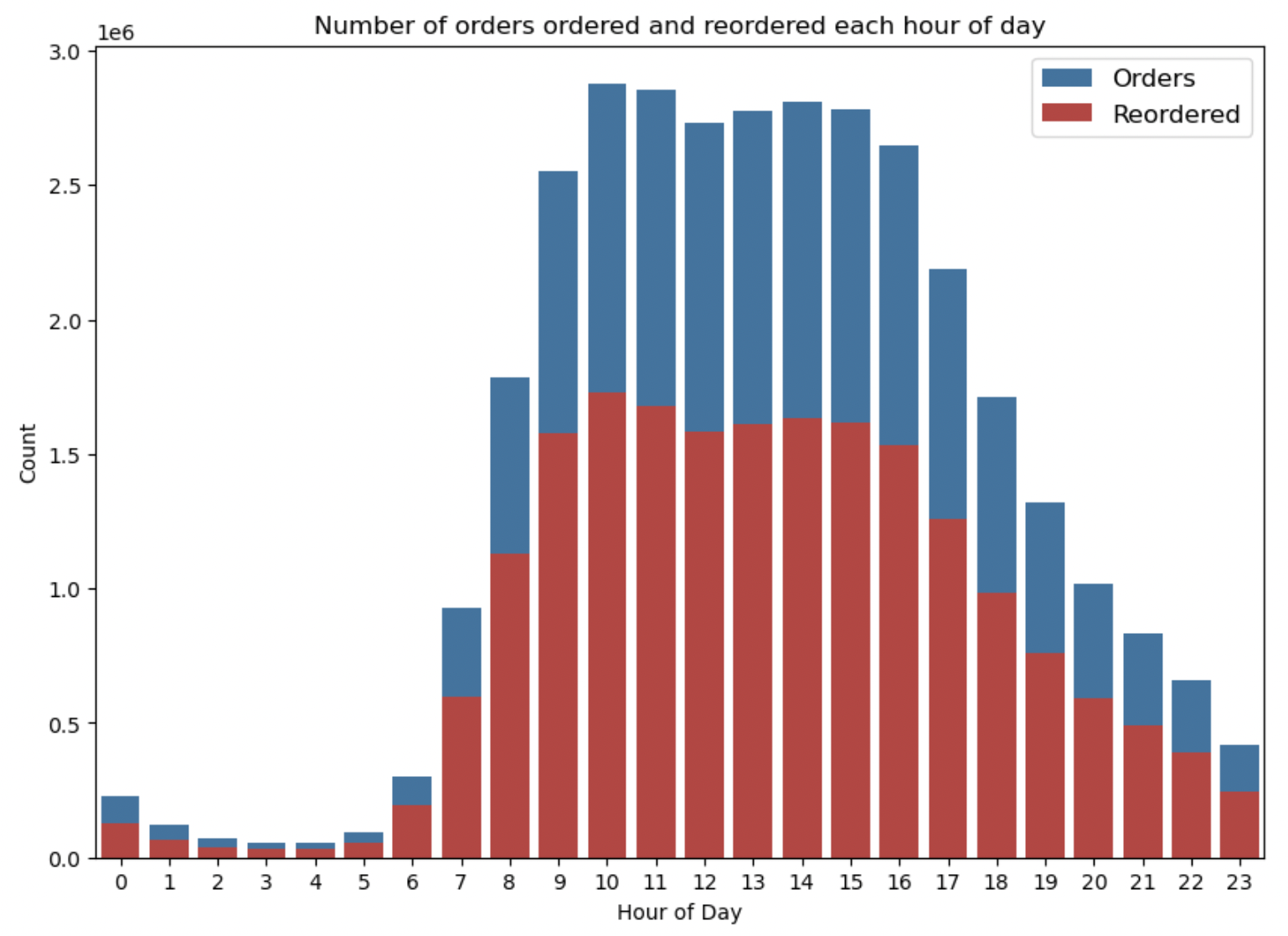
Description automatically generated



* Considering that week starts on Sunday most orders and reorders were made on Sunday and Monday. Least orders and reorders were made in the middle of the week, i.e., Wednesday and Thursday. Based on the graph we can say that customers tend to restock at the start of the week.
* Orders and reorders are proportionally same for all the days of the week.

**Q8. Number of products ordered and reordered in a particular hour of the day.**





* From the graph above it can be inferred that most of the orders and reorders were made from morning to midnight with higher frequency between 9 AM to 5 PM. While very few orders and reorders were made from midnight to early morning.

**Feature Engineering:**

**Merging Data**

First, we merge prior product order data with orders. The resultant DataFrame contains some important columns which will be useful when creating new features.

**Feature Creation**

* Firstly, we create product-level features as follows:

1. **Product\_bought\_rate -** This feature denotes the rate at which a product was bought by a user.
2. **Product Features:**
   1. **prod\_add\_to\_cart\_order\_mean -** Average position of a product in the cart w.r.t mean of each value in add\_to\_cart\_order.
   2. **prod\_order\_total -** This is the total number of times a product was ordered by calculating total count of a product in reorder column.
   3. **prod\_reorder\_total -** Total number of times a product was reordered by calculating sum of times product was reorder.
   4. **prod\_reorder\_rate -** Reorder rate of a product by calculation avg of reorders for each product.
   5. **prod\_unique\_users -** Total number of unique users by getting unique occurrences of each user.
   6. **first\_order\_total -** Product buying rate for the first time by calculating sum of product\_bought\_rate where it is equal to 1
   7. **second\_order\_total -** Product buying rate for the second time by calculating sum of product\_bought\_rate where it is equal to 2.
   8. **is\_organic -** Check whether a product is organic or not by inputting 1 if product has ‘organic’ else 0.
3. **Aisle Features:**
   1. **aisle\_add\_to\_cart\_order\_mean -** Aisle cart position by calculating add\_to\_cart\_order mean for each aisle.
   2. **aisle\_add\_to\_cart\_order\_std -** Aisle cart std by calculating add\_to\_cart\_order std for each aisle.
   3. **aisle\_order\_total -** total orders made for each aisle by calculating count of all reorders for each aisle.
   4. **aisle\_reorder\_total -** total reorders made for each aisle by calculating sum of all reorders for each aisle.
   5. **aisle\_reorder\_rate -** reorder rate for each aisle by calculating mean of all reorders for each aisle.
   6. **aisle\_unique\_users -** unique users for each aisle by fetching unique values in user\_id column.
4. **Department Features:**
   1. **dept\_add\_to\_cart\_order\_mean -** Department cart position by calculating add\_to\_cart\_order mean for each department.
   2. **dept\_add\_to\_cart\_order\_std -** Department cart std by calculating add\_to\_cart\_order std for each department.
   3. **dept\_order\_total -** total orders made for each department by calculating count of all reorders for each department.
   4. **dept\_reorder\_total -** total reorders made for each department by calculating sum of all reorders for each department.
   5. **dept\_reorder\_rate -** reorder rate for each department by calculating mean of all reorders for each department.
   6. **dept\_unique\_users -** unique users for each department by fetching unique values in user\_id column.

* In the next step, we merge all the new features data frames into one product features data frame.
* Next remove the following unnecessary fields from the data frame - 'product\_name', 'aisle\_id', 'department\_id'.
* Then encode 'aisle\_id' & 'department\_id' columns using category\_encoders module’s BinaryEncoder.
* Next, create user-level features as follows:

1. **dow\_mean -** order day of week avg for each user by calculating mean of order\_dow.
2. **dow\_std -** order day of week std for each user by calculating std of order\_dow.
3. **doh\_mean -** order hour of day avg for each user by calculating mean of order\_hour\_of\_day.
4. **doh\_std -** order hour of day std for each user by calculating std of order\_hour\_of\_day.
5. **days\_since\_prior\_order\_mean -** number of days since prior avg for each user by calculating mean of days\_since\_prior\_order.
6. **days\_since\_prior\_order\_std -** number of days since prior for each user by calculating std of days\_since\_prior\_order.
7. **unique\_orders\_by\_user -** unique orders for each user by fetching unique values in order\_number column.
8. **unique\_product\_by\_user -** unique products for each user by fetching unique values in product\_id column.
9. **products\_by\_user -** totat products bought by a user by calculating total count.
10. **reorders\_by\_user -** total reorders made by each user by calculating sum of all reorders for each user.
11. **reorder\_rate\_by\_user -** reorder rate for each user by calculating mean of all reorders for each user.
12. **order\_size\_avg -** User's average size of order by calculating total count of reorders on data grouped by user\_id & order\_number and then grouped by user\_id of the new data frame.
13. **reorder\_in\_order -** User's mean of all orders of reorders by calculating total count of reorders on data grouped by user\_id & order\_number and then grouped by user\_id of the new data frame.
14. **orders\_1, orders\_2, orders\_3 -** total number of order items in the user's last three orders.
15. **reorder\_1, reorder\_2, reorder\_3 -** reorder rate in the user's last three orders.

* Next, create user product features as follows:

1. **total\_orders\_by\_user -** Total orders by a user for a product by calculating the count of all reorders on data grouped by user\_id & product\_id.
2. **total\_reorders\_by\_user -** Total reorders by a user for a product by calculating the sum of all reorders on data grouped by user\_id & product\_id.
3. **user\_reorder\_rate -** Reorder rate of a user for a product by calculating the mean of all reorders on data grouped by user\_id & product\_id.
4. **add\_to\_cart\_by\_user\_mean -** Cart order of a user for products by calculating the mean of add\_to\_cart\_order on data grouped by user\_id & product\_id.
5. **days\_since\_prior\_order\_avg -** Average days since the prior order of a user for products by calculating the mean of days\_since\_prior\_order on data grouped by user\_id & product\_id.
6. **product\_last\_bought\_order -** order number when the product was last ordered on data grouped by user\_id & product\_id.
7. **is\_reorder\_3, is\_reorder\_2, is\_reorder\_1** - Product in last three orders.

* Now we must merge train data with orders data and remove the following unnecessary columns - eval\_set, add\_to\_cart\_order, order\_id.
* Filter user product features for all unique users in train data.
* Impute null values with mean values for the following columns - order\_number, order\_dow, order\_hour\_of\_day, days\_since\_prior\_order.
* Then remove orders which were purchased for the first time in the last three orders.
* Finally, merge product and user features with the final features data.
* Based on our previous analysis using models we found there is an overfitting issue. In order to fix this problem, we have removed the correlation between features. By doing so we were able to reduce the total number of feature columns to 52.

**Modelling strategies:-**

The problem at hand is a supervised binary-classification type problem. Using the extracted features (as demonstrated in ‘Feature Engineering’ section), we prepare a dataframe which shows all the products that user has bought previously, user level features, product level features, user-product level features, aisle and department level features, and the information of current order such as order's day-of-week, hour-of-day, etc. We have to use the order details which have ‘eval\_set’ value marked as ‘test’, as our test data. The target variable would be 'reordered' which shows how many of the previously purchased items, user ordered this time.

Below are some noteworthy points which would drive the approach for designing the machine learning models and training them:

1. The products that a user has never purchased can never by reordered. So, we don’t need to worry about the products which a user has never purchased.
2. Only the products which a user has purchased in the past (i.e., prior orders) can be reordered.
3. We have to classify the data into 2 categories, i.e., will a product be reordered or not, where ‘reordered’ is the target variable.
4. Independent variables (input) will contain user, order, product information.
5. Since the provided dataset contains imbalanced data, accuracy score won’t be able to depict model performance efficiently. We would need F1-score as the performance evaluation metric for our models.
6. Since recommending some products which a customer is not going to reorder will lead to the customer exploring new products, a few False Positives doesn't harm.
7. False Negatives for highly reordered products can be costly.

We shall use below machine learning models to solve the classification problem:

1. Logistic regression
2. Decision tree classifier
3. Random Forest classifier
4. XGBoost classifier

**Logistic Regression:-**

This algorithm is widely used when the classification problem at hand is binary. It uses the sigmoid function to return the probability of a label. We have employed feature engineering to gather some critical information regarding order details and used same in model training. Some important hyperparameters for such a model are *penalty, max\_iter, C, solver, etc.*

Since the dataframe is huge (contains ~8.5 million rows), we had to reduce the memory consumption of it by sampling 50% of the data. Also, in our final dataset, we encounter huge class imbalance.

|  |  |  |
| --- | --- | --- |
| **Output variable (‘reordered’)** | **Data count** | **Percentage** |
| 0 | 7,645,837 | 90.21% |
| 1 | 828,824 | 9.79% |

To balance the data, we have used cost-sensitive learning by assigning class weightage ({0:1, 1:10}). We won’t use random-up-sampling/SMOTE as it would increase the data size and also the model training time. Also, since random-down-sampling discards information randomly, we might lose some important information and would result in bias.

* First iteration: We started by training the model on complete dataset which had huge class imbalance. Due to this, our model became biased towards the majority class, i.e., it predicted output as 0 for all input data. As a result, model accuracy for category 0 (not reordered) raised to 90%, but for category 1 (reordered) it became NILL.
* Optimization: We observed that model was overfitting and to remove this overfitting issue, we eliminated the collinear features from the dataset. We used the parameter 'class\_weight' in the model estimator and set it to {0:1, 1:10} to tackle class imbalance issue. Also, we reduced the training dataset to train the model in stipulated time, while still keeping all the information from data. We also used ‘GridSearchCV’ method to cross-validate different values for parameters such as 'C':[4,5,6], and used the best fit set to optimize the model.
* Final observation: Post optimizing the model and training it over multiple combinations, we observed below results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Precision** | **Recall** | **F1-Score** |
| **0** | 0.97 | 0.57 | 0.72 |
| **1** | 0.17 | 0.81 | 0.28 |

|  |  |
| --- | --- |
| **Metric** | **Score** |
| **Accuracy** | 0.60 |
| **F1-Score** | 0.28 |

**Decision Tree classifier:-**

This is one of the easiest and popular classification algorithms. Decision Tree requires relatively less data preparation and has low computational cost. It is capable of handling both numerical and categorical data, which is in line with our dataset. Contrarily, Decision trees can have some variance, which can be lowered by methods like boosting. Decision trees are prone to overfitting, especially when a tree is particularly deep. One way to combat this issue is by setting a max depth. This will limit our risk of overfitting; but as always, this will be at the expense of error due to bias. Pruning refers to a technique to remove the parts of the decision tree to prevent growing to its full depth. To prevent Decision Tree classifier from overfitting, we use post-prunning and pre-prunning techniques. This gives a simpler model with less variance sample-to-sample but ultimately will not be a strong predictive model. Ideally, we would like to minimize both error due to bias and error due to variance. Decision tree algorithm is a white-box model as we can determine how each feature (node) is splitting up.

* First iteration: We started by training the model on complete dataset which had huge class imbalance. Due to this, our model became biased towards the majority class, i.e., it predicted output as 0 for all input data. As a result, model accuracy for category 0 (not reordered) raised to 90%, but for category 1 (reordered) it became NILL.
* Optimization: We observed that model was overfitting and to remove this overfitting issue, we eliminated the collinear features from the dataset. We used the parameter 'class\_weight' in the model estimator and set it to {0:1, 1:10} to tackle class imbalance issue. Also, we reduced the training dataset to train the model in stipulated time, while still keeping all the information from data. We also used ‘GridSearchCV’ method to cross-validate different values for parameters such as 'max\_depth': [3,4,5,10,15], 'max\_features': ['auto'], ‘min\_samples\_split': [2,3,4,5], 'class\_weight': [{0:1, 1:10}] and used the best fit set to optimize the model.
* Final observation: Post optimizing the model and training it over multiple combinations, we observed below results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Precision** | **Recall** | **F1-Score** |
| **0** | 0.95 | 0.80 | 0.87 |
| **1** | 0.24 | 0.58 | 0.34 |

|  |  |
| --- | --- |
| **Metric** | **Score** |
| **Accuracy** | 0.77 |
| **F1-Score** | 0.33 |

**Random Forest classifier:-**

This is a collection of multiple Decision Trees, and is amongst the most powerful Machine learning algorithms today. It takes less training time and predicts output with high accuracy. It runs efficiently even for large datasets, and prevents the overfitting issue. In case of Classification problem, majority voting is used to determine final output from all the underlying decision trees. There is no need to perform data Normalization. Normalization is particularly important in algorithms that are distance-based, such as KNN and K-means as it requires Euclidean Distance. However, the Random Forest algorithm is not a distance-based model - it is a tree-based model. Each node in a Random Forest is not comparing feature values, it is simply splitting a sorted list that requires absolute values for branching. To enhance predictive power of model we can tune some important hyperparameters such as n\_estimators, max\_features, max\_depth, min\_samples\_split, min\_samples\_leaf, etc. Random Forest classifier is not affected by outliers. This is because it selects features randomly and hence, gives equal importance to all.

* First iteration: We started by training the model on complete dataset which had huge class imbalance. Due to this, our model became biased towards the majority class, i.e., it predicted output as 0 for all input data. As a result, model accuracy for category 0 (not reordered) raised to 90%, but for category 1 (reordered) it became NILL.
* Optimization: We trained the model by eliminating the collinear features from the dataset, and observed slight improvements in the prediction score. We used the parameter 'class\_weight' in the model estimator and set it to {0:1, 1:10} to tackle class imbalance issue. Also, we reduced the training dataset to train the model in stipulated time, while still keeping all the information from data. We also used ‘RandomizedSearchCV’ method to perform cross-validation with each combination of parameter values such as 'max\_depth': [10,15,20,25], 'min\_samples\_split': [1,2,3], 'class\_weight': [{0:1, 1:10}] and used the best fit set to optimize the model.
* Final observation: Post optimizing the model and training it over multiple combinations, we observed below results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Precision** | **Recall** | **F1-Score** |
| **0** | 0.95 | 0.84 | 0.89 |
| **1** | 0.30 | 0.63 | 0.41 |

|  |  |
| --- | --- |
| **Metric** | **Score** |
| **Accuracy** | 0.82 |
| **F1-Score** | 0.40 |

**XGBoost classifier:-**

This is an implementation of Gradient Boosted decision trees, wherein decision trees are created in sequential form. We can achieve higher model performance using ‘GridSearchCV’ by tweaking some crucial parameters such as learning\_rate, gamma, reg\_alpha, reg\_lambda, max\_depth, subsample, etc. This ML algorithm performs extremely well with large, structured datasets containing too many features, making it a perfect choice for our problem statement. For faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. As a result, model training with large datasets takes lesser time as compared to some other algorithms. It uses advanced regularization (L1 & L2) techniques, which improves model generalization and eliminates the overfitting problem. Extreme Gradient Boosting algorithm is more robust to outliers than some other boosting algorithms like AdaBoost, Gradient Boost, etc.

* First iteration: We started by training the model on complete dataset which had huge class imbalance. Since XGBoost algorithm handles class imbalance inherently better than some other models, we got a better initial score when compared to previously tested models. But the score was still not satisfactory.
* Optimization: We trained the model by eliminating the collinear features from the dataset, and observed slight improvements in the prediction score. We used the parameter "scale\_pos\_weight" in the model estimator and set it to 10 to tackle class imbalance issue. We also used ‘*GridSearchCV’* method to perform hyper-paramter tuning on multiple combinations of parameter values. Then we got the best parameter values as "objective”: "reg:logistic", "eval\_metric": "logloss", "eta": 0.1, "max\_depth": 6, "min\_child\_weight":10, "gamma": 0.70, "subsample": 0.76, "colsample\_bytree": 0.95, "alpha": 2e-05, "scale\_pos\_weight": 10, "lambda":10 and used this best fit set to optimize the model.
* Final observation: Post optimizing the model and training it over multiple combinations, we observed below results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Precision** | **Recall** | **F1-Score** |
| **0** | 0.97 | 0.74 | 0.84 |
| **1** | 0.24 | 0.77 | 0.37 |

|  |  |
| --- | --- |
| **Metric** | **Score** |
| **Accuracy** | 0.74 |
| **F1-Score** | 0.37 |

**Result and analysis:-**

We performed feature engineering to extract some critical information from the provided dataset. Then we fed this comprehensive dataset to our machine learning models and performed model training. We performed hyper-parameter tuning and class-balancing by increasing the weight of minority class in our dataset. We started with the most basic machine learning algorithm for classification use-case, i.e., ‘Logistic Regression’. As this model was overfitting our dataset, we quashed multi-collinearity by removing some highly collinear features. Then, we moved to train ‘Decision tree classifier’ on our dataset. Decision trees are also prone to overfitting, so we performed pre-prunning to overcome this trouble. This technique gives a simpler model with less variance but ultimately generates a weaker predictive model. Ideally, we would like to minimize both error due to bias and error due to variance. Random forests mitigate this problem well. So, we trained ‘Random Forest classifier’ on our dataset. Using this model, we were able to predict both the class outcomes comprehensively, without overfitting the model. Finally, we used ‘XGBoost classifier’ and trained the same on our complete dataset. XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data. As a result, it handles sparse dataset very efficiently and also achieves high computation speeds.

Below are the cumulative performance metrics scores for all the employed ML models:-

|  |  |  |
| --- | --- | --- |
| **ML model** | **Accuracy score** | **F1-Score** |
| Logistic regression | 0.60 | 0.28 |
| Decision tree classifier | 0.77 | 0.33 |
| Random Forest classifier | 0.82 | 0.41 |
| XGBoost classifier | 0.74 | 0.37 |

1. Logistic regression analysis:

* Recall score in predicting 0 is very less. This means that model is unable to correctly predict true positive values of 0.
* Also, precision score in predicting 1 is very less, which symbolizes that model is unable to predict 1’s efficiently.
* As a result, model overall F1-score and accuracy are substantially low.

Chart, treemap chart

Description automatically generated

1. Decision Tree analysis:

* Precision and recall scores in predicting 0 is high. This means that model is able to correctly predict true positive values of 0.
* Precision score in predicting 1 is very less, which symbolizes that model is unable to predict 1’s efficiently.
* As a result, model overall F1-score and accuracy are better than previous model, but still not satisfactory.

Chart

Description automatically generated

1. Random Forest analysis:

* Precision and recall scores in predicting 0 is very high. This means that model can efficiently predict true positive values of 0.
* Precision score in predicting 1 is high, which symbolizes that model doesn’t give too many false positives.
* Recall score in predicting 1 is less. This means that model is unable to correctly predict all the true positive values of 1.
* As a result, model overall F1-score and accuracy are very high.

Chart

Description automatically generated

1. XGBoost analysis:

* Precision and recall scores in predicting 0 is very high. This means that model can efficiently predict true positive values of 0.
* Precision score in predicting 1 is lesser as compared to Random Forest, which symbolizes that model gives slightly more false positives.
* Recall score in predicting 1 is more. This means that model is able to correctly predict most of the true positive values of 1.
* As a result, model overall F1-score and accuracy are high.

Chart

Description automatically generated

Performance metrics we have used for this use-case are explained below:-

* Accuracy: This is a metric for classification models that measures the number of predictions that are correct as a percentage of the total number of predictions that are made.

A picture containing table

Description automatically generated

Accuracy is not a good metric to use when we have class imbalance. It does not give equal importance to all the classes. So, we will lay more emphasis on other performance metrics.

* F1 score: The F1 score is defined as the harmonic mean of precision and recall, where both precision and recall are given equal importance. A model will obtain a high F1 score if both Precision and Recall are high.

Text

Description automatically generated

**Conclusion:-**

We approached the problem statement by breaking it down into smaller sections. At first the use-case would appear as multi-class classification problem, but we converted into a supervised binary-class classification use-case. Then 4 Machine learning models were built with over 50+ features to achieve the goal. Weighted average of all these probabilities were used to generate recommendation such that they have high F1 Score. Careful and precise analysis was performed over all the aspects of these models. We compared all the performance metrics scores of these models and analyzed the significance of each. Post introspection of all these models, we can narrow down to **‘Random Forest classifier’** as the best performing model for our use-case. Precision and recall scores in predicting both, 0 and 1 are higher for Random Forest algorithm as compared to other models. This means that model can efficiently predict true positive values of 0 and 1. Consequently, the cumulative F1-score for the model is highest. So, we can confidently quote that this model will generate the most efficient predictions in determining whether a product will get reordered by a user in their future order.

**Future Work:-**

* Deploy this application on a remote server using AWS.
* We can implement a solution to this problem using Deep Learning in a more efficient form.
* We can extend this solution to provide even more recommendations, such as for each product we can suggest an item which was most frequently purchased with it.

**References:**

* <https://stackoverflow.com/questions/62581004/how-to-set-class-weights-in-decisiontreeclassifier-for-multi-class-setting>
* <https://github.com/shubhamscifi/Instacart-Market-Basket-Analysis>
* <https://github.com/archd3sai/Instacart-Market-Basket-Analysis>
* <https://github.com/KazukiOnodera/Instacart>
* https://asagar60.medium.com/instacart-market-basket-analysis-part-1-introduction-eda-b08fd8250502