**Market basket Analysis Using Machine learning**

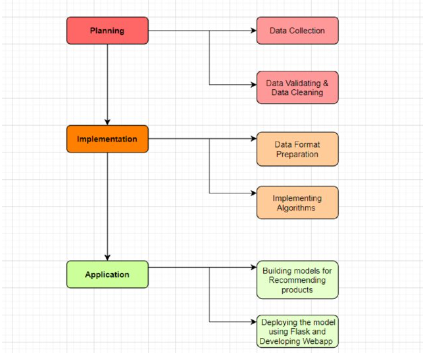
# Project Description:

Ordering food supplies online is a new way of restocking groceries and other essential items. Be it early morning or midnight, ordering groceries online is stress-free activity without much hassle. But what happens when you forget few items while adding items to the cart or want to get better suggestions on your items? Will you wait for a couple of hours and then order? To deal with such situations, users are provided with suggestions based on their past orders or user preferences. Instacart, a grocery order and the delivery app with over 500 Million products and 40000 stores serves across U.S. & Canada. Instacart provides a user experience where you will get product recommendation based on your previous orders. Instacart provided us with transactional data of customer orders over time to predict which previously purchased products will be in a user’s next order. This data is open-sourced and given as a Kaggle challenge.

**Real-World / Business Objectives and Constraints**

The objective 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, Instacart provided between 4 and 100 of their orders, along with the sequence in which products were placed in the cart.

**Technical Architecture Diagram:**



# Pre requisites:

**To complete this project, you must required following software’s, concepts and packages**

* **Anaconda navigator and pycharm:**
  + Refer the link below to download anaconda navigator
  + Link : <https://youtu.be/1ra4zH2G4o0>
* **Python packages:**
  + Open anaconda prompt as administrator
  + Type “pip install numpy” and click enter.
  + Type “pip install pandas” and click enter.
  + Type “pip install scikit-learn” and click enter.
  + Type ”pip install matplotlib” and click enter.
  + Type ”pip install scipy” and click enter.
  + Type ”pip install pickle-mixin” and click enter.
  + Type ”pip install seaborn” and click enter.
  + Type “pip install Flask” and click enter.

# Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

* **ML Concepts**
  + Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  + Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
  + Regression and classification
  + Logistic Regression: [Machine Learning - Logistic Regression (tutorialspoint.com)](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_logistic_regression.htm)
  + Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
  + Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
  + Multi layer perceptron model : [Multi-layer Perceptron in TensorFlow - Javatpoint](https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow)
  + Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
  + Catboost : [How CatBoost Algorithm Works In Machine Learning (dataaspirant.com)](https://dataaspirant.com/catboost-algorithm/)
  + Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
* **Flask Basics** : <https://www.youtube.com/watch?v=lj4I_CvBnt0>

# Project Objectives:

By the end of this project you will:

* Know fundamental concepts and techniques used for machine learning.
* Gain a broad understanding about data.
* Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.
* Understand the application of the machine learning algorithms applied
* Understand the working of market basket analysis used in recommendation systems

# Project Flow:

* User interacts with the UI to enter the input.
* Entered input is analyzed by the model which is integrated.
* Once model analyses the input the prediction is showcased on the UI
* Similar product reccomendations are provided on the basis of the existing products purchased in the cart

Work flow

* Data collection
  + Collect the dataset or create the dataset
* Visualizing and analyzing data
  + Univariate analysis
  + Bivariate analysis
  + Descriptive analysis
* Data pre-processing
  + Checking for null values
  + Handling outlier
  + Handling categorical data
  + Splitting data into train and test
* Model building
  + Import the model building libraries
  + Initializing the model
  + Training and testing the model
  + Evaluating performance of model
  + Save the model
* Application Building
  + Create an HTML file
  + Build python code

**STEP 1:** **Dataset and explanation of features**

Step 1: Downloading the Dataset

The dataset for this project is downloaded from kaggle. The link for the same is provided as follows:

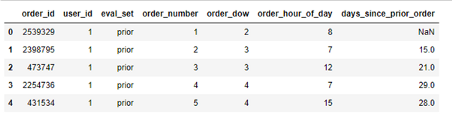
<https://www.kaggle.com/c/instacart-market-basket-analysis>

**Explanation Of The Dataset features**:

Data can be broadly divided into 3 parts.

* Prior data : Order history of every user . This data contains nearly 3–100 past orders per user
* Train data : Current order data of every user . This data contains only 1 order per user
* Test data : Future order data of every user . This data will not contain any product information ( We need to predict that )

**orders.csv** — consists of order details placed by any user — shape: (3421083, 7)



data snippet for orders.csv

* Order\_id : Unique for every order
* User\_id : Unique for every user
* Eval\_set : ( prior / train / test)
* Order\_number : ith order placed by user
* Order\_dow : Day of week
* Order\_hour\_of\_day : Time of day in hr
* Days\_since\_prior\_order : difference in days between 2 orders

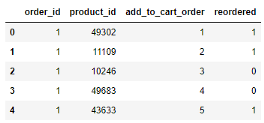
**order\_products\_\_prior.csv —**consists of all product details for any prior order, shape: (32434489, 4)



data snippet for order\_products\_prior.csv

* order\_id : Unique order id for every order
* product\_id : product ID of item
* add\_to\_cart\_order : denotes the sequence in which products were added to cart.
* reordered : product is reordered ? (1/0)

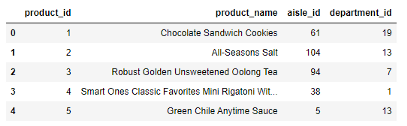
**order\_products\_\_train.csv —**consists of all product details for a train order, shape: (1384617, 4)



data snippet for order\_products\_train.csv

* order\_id : Unique order id for every order
* product\_id : product ID of item
* add\_to\_cart\_order : denotes the sequence in which products were added to cart.
* reordered : product is reordered ? (1/0)

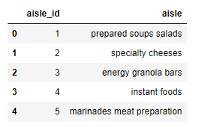
**products.csv —**details of a product, shape: (49688, 4)



data snippet for products.csv

* product\_id : product ID of item
* product\_name : name of product
* aisle\_id : aisle id of the product
* department\_id : department id of the product

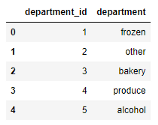
**Aisles.csv —**details of aisles, shape: (134,2)



data snippet for aisles.csv

* aisle\_id : aisle ID of item
* aisle\_name : name of aisle

**department.csv —**details of department, shape: (21,2)



data snippet for department.csv

* department\_id : product ID of item
* department\_name : name of department

# STEP 2: Approach towards solving the problem

# Based on these orders history and user preferences of the product, we need to predict the products which could be reordered.

**Type of Machine Learning Problem**

At first, it seems like Multi-Label Classification, but there are 49688 products, and total product recommendations could be anywhere from None to N. Therefore, this problem is restructured into a binary classification problem, where we will predict the probability of an item being reordered by a user.

For each order, we will group these probabilities to pick top K probable products which will be reordered, and recommend those to the user.

**Cost Function**

Since we have converted this to a binary classification problem, we will use logloss as cost function, as we want the model to penalize incorrect classification and want high probability of correct class.

https://miro.medium.com/max/351/1*tYQ8TgjpwVAmNiHey6BQqg.jpeg

Log loss Equation

**Performance Metric**

Mean F1-Score — Mean of all F1 Scores for every order.

Since we need to know how many of the actual recommended products match with predicted ones, we will use F1 score on each order. For all orders combined, we will take mean of those F1-scores.

# Project Structure:

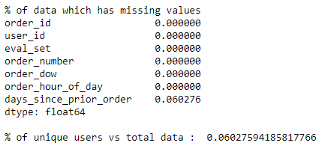
* We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
* Model.pkl is our saved model. Further we will use this model for flask integration.
* Training folder contains model training files and training\_ibm folder contains IBM deployment files.

STEP 3: Exploratory Data Analysis

EDA is important as it helps to answer many questions about the data which further leads to better feature engineering.

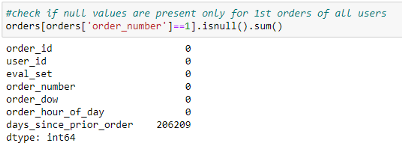
**Presence of Missing Values**

The dataset has no missing values except for in orders.csv



percentage of empty cells : As we can see only 6% of values are missing from days\_since\_prior\_order column.

Also there are 6% of unique users (206209) compared to total data in orders.csv



Nan values for order\_number =1: It can be seen that for every user’s 1st order ( order\_number = 1) the days\_since\_prior\_order is Nan, which makes sense. We can impute 0 here.

**Merge Data**

We will merge all order\_products\_\_prior.csv, order\_products\_\_train.csv, products.csv, orders.csv ( only train and prior set), aisle.csv and departments.csv for perform EDA on**.**

**Step 4: Visualizing and Analyzing Data**

**Univariate Analysis**

**Q : What is distribution of Target variable ?**

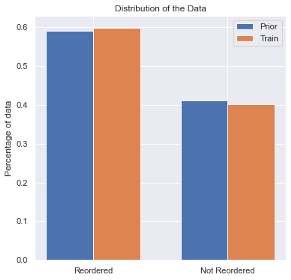
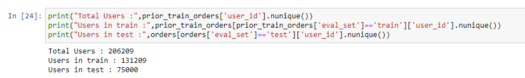


Fig:Target Data Distribution

Conclusions:

* Distribution is similar in both Prior and Train Set
* Around 60% of time product has been reordered

## Q : How many users are there ?



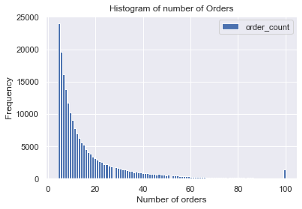
user distribution

* Every User in Test set has prior orders in order\_products\_prior.csv, similarly for every user in Train has order history in order\_products\_prior.csv.
* So , we can conclude the train-test data is split on users

## Q : How many orders were placed by every user ?

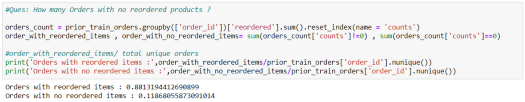


Fig:Orders placed by user



* For every user we have around 4–100 order details ( including train and test)
* Histogram on left shows that there are very few users who have placed orders more than 60 , and maximum order for any user is 100.

## Q: How many Orders with no reordered products ?



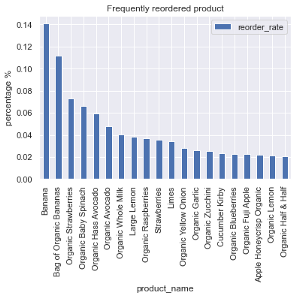
Orders with no reordered products

* 12 % of orders have no reordered items, while rest ~88 % of orders contains reordered items

**Q: Which are the most frequently ordered / reordered products ?**

https://miro.medium.com/max/525/1*7a0HHsLTTY2dzL9ooGxOfA.png

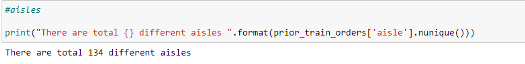
Total Products



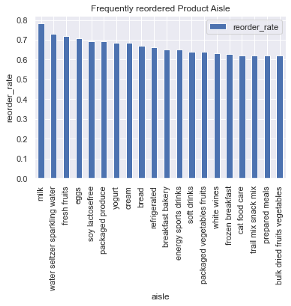
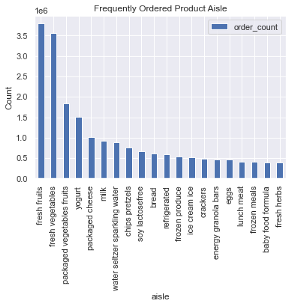
Frequently reordered products

* There are total 49685 products which were ordered.
* It can be seen that most of the products which are ordered are organic foods / fresh fruits (especially Bananas)
* Bananas have highest order rate of 0.14.
* Top 5 frequently ordered products are organic in nature (This could be an important feature)

**Q : How frequently products are ordered and reordered from each aisles ?**



Total Aisles

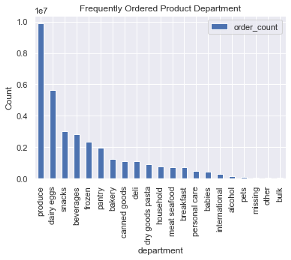
Frequently ordered / reordered product aisle

* There are total 134 different aisles
* As we can see, most products are ordered from Fresh Fruits and Fresh Vegetables aisles.
* Other frequently ordered items are from Yogurt , Packaged Vegetables and packaged cheese aisles.
* Least frequently ordered items are from Air fresheners, Baby accessories, Baby bath body care etc. aisles
* Milk, sparkling water, fruits, eggs, yogurt are most common aisles the product is reordered from, as they are items which are daily consumed, and one rarely switches from their usual meal plan. Also these are the products that lasts only few days , thus high reorder rate.
* On the other hand hair care, skin care, kitchen supplies are the one which lasts longer than other, hence low reorder rate.

**Q : How frequently products are ordered and reordered from each departments ?**

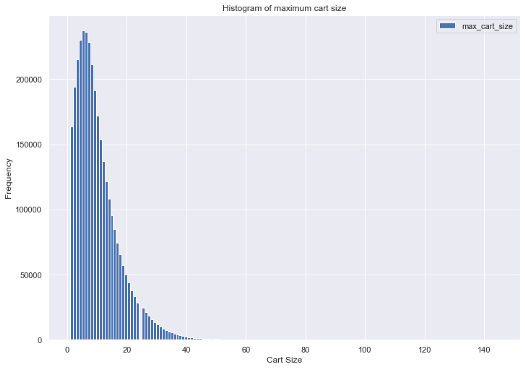
https://miro.medium.com/max/525/1*shqgTWQ6RnC5lA1-4NAXzA.png

Total departments

Frequently ordered / reordered product department

* There are total 21 departments
* As seen from departments analysis , most ordered products are from produce department which have fresh vegetables, fruits, herbs etc. But most reordered product department is dairy eggs having yogurt, milk, eggs, cheese etc.
* We see high reorder rate in organic foods and daily consumed items.
* Low reorder rate in personal care departments

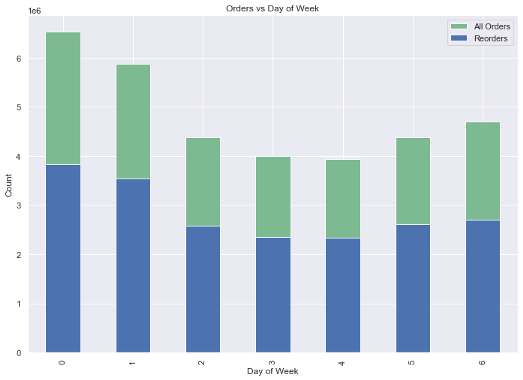
**Q : What is the cart size on different orders ?**



Histogram of cart size

* We have a right skewed distribution of maximum cart size for every order
* There are 237225 orders with cart size = 5, also, mode = 5.
* There are very few order with cart size > 40 and all the way up to 145.

**Q : How many products were ordered and reordered on a particular day of week ?**

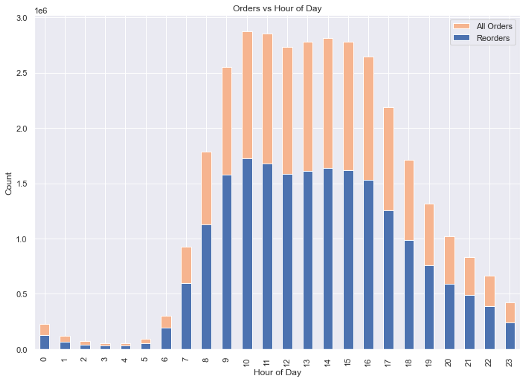


O r d e r s v s D a y o f w e e k

Assuming that the week starts from Sunday, most shopping is done on Sundays and Mondays. Also least orders were placed on Thursday. People tend to restock there supplies on Sundays.

* Reorders w.r.t to days of week is proportionally same as all orders.

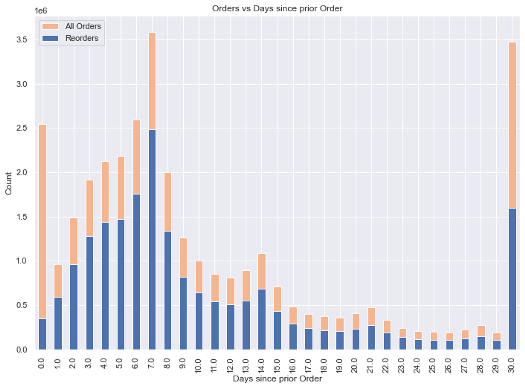
**Q : How many products were ordered and reordered on a particular hour of day ?**



Orders vs Hour of Day

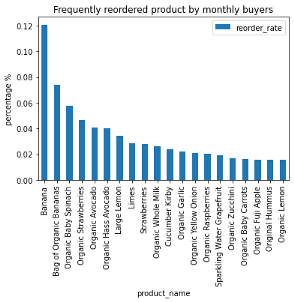
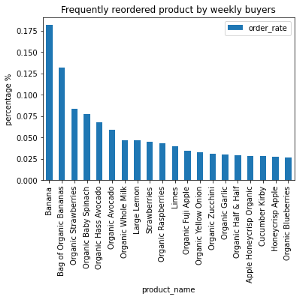
* Most orders are placed from Early morning to midnight, and very few orders placed from midnight to early morning.

**Q : After how many days user ordered / reordered a product ?**



* Most people restock after a week or a month. It seems, some people prefer buy a week / month supplies at once.
* People who are buying at Day 0, are probably new customers, but we can see a small rate of reorder implying that users tend to place multiple orders on Day 0 too.
* Probably here 30 days represents the upper limit, and not necessarily any particular month.
* There is a continuous spike in orders from day 1 to day 6, shows that some people are frequent buyers with short window of restocking.

**Q : Which products are frequently brought by weekly buyers and monthly buyers ?**

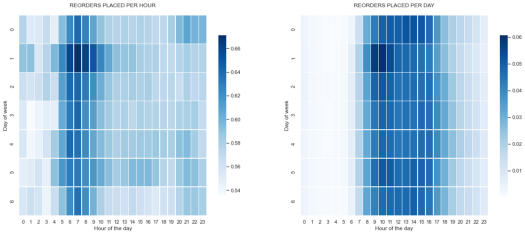


Frequently reordered product by weekly and monthly buyers

* Weekly/monthly buyers tend to buy similar products
* These products, in general, have highest reorder rate irrespective of day of purchase

## **Bivariate Analysis**

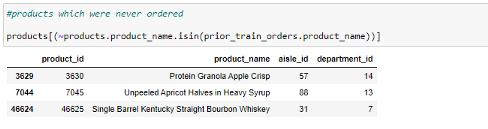
## Q : How day of week and hour of day impact product order / reorder ?



* first plot describes reorder rate of every day w.r.t to orders placed at that hour
* second plot describes reorder rate of every day w.r.t to orders placed on that day
* from first plot we can see that of all orders that were placed on any hour, most reorders were placed on day 1 ( probably Monday) from 5 AM — 9 AM.
* Same pattern can be seen on any day between 5 AM — 9 AM.
* from second plot we can see that of all orders that were placed on any day, most reorders were placed from 8 AM — 4 PM, on any given day

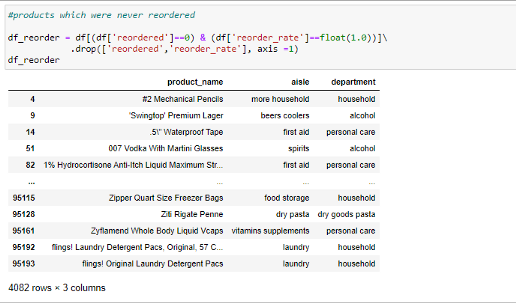
**Other Insights**

**Q : Which products were never ordered / reordered?**



Products which were never ordered

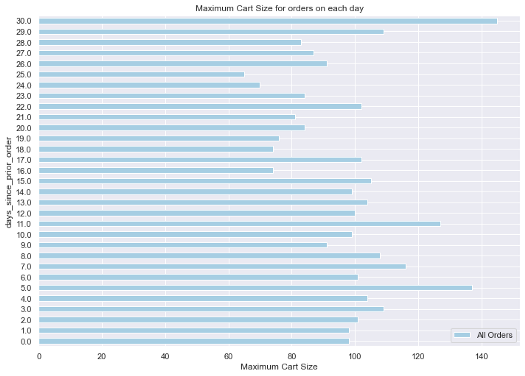
These 3 products (Protein Granola Apple Crisp, Unpeeled Apricot Halves in Heavy Syrup , Single Barrel Kentucky Straight Bourbon Whiskey) which were never ordered. May be their alternatives were ordered.



Products which were never reordered

* Of all 49685 products 4082 products were never reordered.
* Products which were never reordered, includes very specific items such as “.5” Waterproof Tape”, “007 Vodka With Martini Glasses”, ‘Swingtop’ Premium Lager”, we can assume that either consumer didn’t like them or he switched his preferences. Looking at household items like “Zipper Quart Size Freezer Bags”, “flings! Laundry Detergent Pac’s, Original”, we cant definitely say that preferences were switched, since they are type of products which lasts long and consumer might consider reordering them in future.

## Q : Maximum cart size after N days since prior order ?



Max cart size vs days since prior order

* As it was expected, users with 30 days gap between consecutive orders have maximum cart size. i.e. they tends to order more products
* Same observation can be seen with 29 days, 11 days, 5 days gap also.
* Users placing order on Day 0, have maximum cart size of around 100, these include both first time orders and multiple orders
* Also, average cart size for orders on any day is around 8.

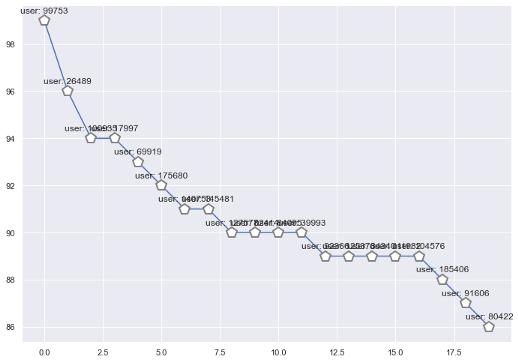
## Q: How much position of product in the cart impact reorder rate?



impact of product position in the cart on reorder rate

* We see a normal decrease in reorder probability when the position of product is increased till 71.
* But probability fluctuates rapidly after position is increased from 71.
* The lowest reorder probability is somewhere around 0.2 when position is 109
* Position from 128 to 137 shows continuous 50 % reorder probability.
* We can assume that for a product with position greater than 100 have very low probability of being reordered (below 0.3)
* Since the reorder probability of a product will depend on product position in the cart and product itself, the vague behavior can be assumed to be the result of reshuffled position of a product with high reorder probability.

## Q: Are there any users whose order contains only reordered products ?



users with orders containing only reordered products

* User\_id 99753 have 99 orders which contains only reordered items
* Followed by User 26489 and 100935

Now that we have completed our EDA , we can move on to next stage

# ****Step 5: Modelling Strategy****

**Strategy : 1**

**Generate Training Data** (using prior\_orders\_data )

* For every orders in prior\_orders\_data, take n-1 orders of every user for feature engineering.
* nth order of every user will used to label the dependent variable i.e. reordered.

example : —

let, user A have 90 orders in prior\_orders\_data.

* build features using 89 orders.
* based on these features we will label the data with reordered(0/1) if any of the products he brought in 89 orders appeared in his 90th order.

**Generate Validation Data** ( Using train\_orders\_data)

* Now that our training data is generated using prior\_orders\_data, we could leverage train\_orders\_data ( which contains 1 order per user) to test our trained model
* we will predict the product reorder probability using trained model to give accuracy.
* Then we will pick top probable products whose probability of reordering was high and which can maximize F1-Score.
* We will use faron’s f1 optimization [code](https://www.kaggle.com/mmueller/f1-score-expectation-maximization-in-o-n) to do this.
* check the actual F1 score , and calculated F1 score. This will give us an idea on how effective the model is .

**Generate Test Data** ( from orders.csv with eval ==’test’)

* Add features built on training data , based on orders and users
* For every order and product predict if it is reordered(0/1)
* Then we will pick top probable products whose probability of reordering was high and which can maximize F1-Score.
* We will use faron’s f1 optimization [code](https://www.kaggle.com/mmueller/f1-score-expectation-maximization-in-o-n) to pick products maximizing f1 score.

**Strategy : 2**

**Generate Training Data** (using prior\_orders\_data and train\_orders\_data)

* Build features on prior\_orders\_data
* Order from train\_orders\_data for every user will be used to label the dependent variable i.e. reordered.
* we will predict the product reorder probability .
* Then we will pick top probable products whose probability of reordering was high

**Step 6: Feature engineering**

we want to predict if , User A →will buy Product B →in his next order C → reordered(1/0) ?

https://miro.medium.com/max/525/1*R5QGkMSFAmge3X61Sg0F9g.png

Dataset structure

This Future Order ID is obtained from train orders and test orders in orders.csv.

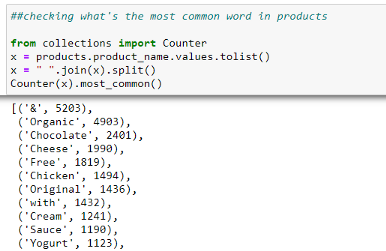
This structure is inspired by **Symeon Kokovidis ‘s**[**kernel**](https://www.kaggle.com/kokovidis/ml-instacart-f1-0-38-part-two-xgboost-f1-max)

## **Generate product only features**

* feat\_1 : product\_reorder\_rate : How frequently the product was reordered regardless the user preference ?
* feat\_2 : average\_pos\_incart : Average position of product in the cart ?

next 3 values are calculated based product being

* isorganic
* isYogurt — aisle
* produce — department
* isFrozen — department
* isdairy — department
* isbreakfast — department
* issnack — department
* isbeverage — department



out of 49688 products 4903 products are organic

These features are picked as they were most reordered product type / aisle/ dept. Now these values are then reduced to 3 columns using Non-Negative Matrix Factorization, to reduce sparsity

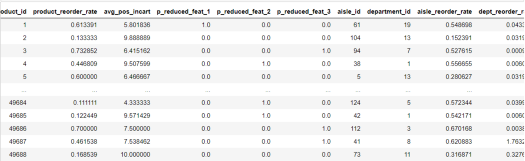
feat\_3 : p\_reduced\_feat\_1 : column 1 from NMF output

feat\_4 : p\_reduced\_feat\_2 : column 2 from NMF output

feat\_5 : p\_reduced\_feat\_3 : column 3 from NMF output

feat\_6 : aisle\_reorder\_rate : How frequently a product is reordered from the aisle to which this product belongs

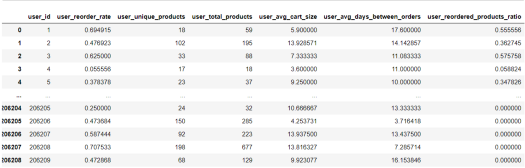
feat\_7 : department\_reorder\_rate : How frequently a product is reordered from the department to which this product belongs



product only features

**Generate User only Features**

* feat\_1 : user\_reorder\_rate : What is the average reorder rate on orders placed by a user ?
* feat\_2 : user\_unique\_products : What is the count of distinct products ordered by a user?
* feat\_3 : user\_total\_products : Count of all products ordered by a user?
* feat\_4 : user\_avg\_cart\_size : Average products per order by a user ? = average cart size ?
* feat\_5 : user\_avg\_days\_between\_orders : Average number of days between 2 orders by a user ?
* feat\_6 : user\_reordered\_products\_ratio : number of unique products reordered / number of unique products ordered

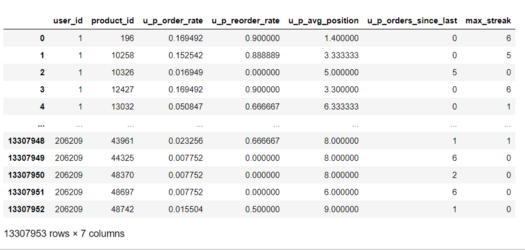


User only features

**Generate User Product Features**

Now that we have created product only and user only features , we will now create features based on how user interacts with a product

* feat\_1 : u\_p\_order\_rate : How frequently user ordered the product ?
* feat\_2 : u\_p\_reorder\_rate : How frequently user reordered the product ?-
* feat\_3 : u\_p\_avg\_position : What is the average position of product in the cart on orders placed by user ?
* feat\_4 : u\_p\_orders\_since\_last : What’s the number of orders placed since the product was last ordered ?
* feat\_5 : max\_streak : Number of orders where user continuously brought a product without miss



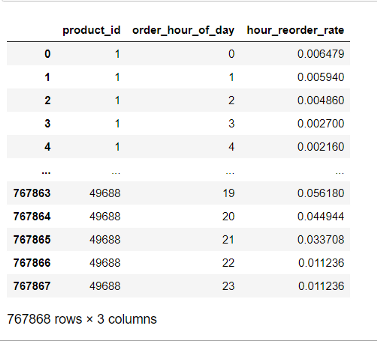
user product features

**Merge above features :**

Now merge these independent features ( user only features , product only features, and user — product features) → call it as merged\_df .Now , this dataframe will contain feature for all possible user — product pairs, some of which will be used in training models (using train orders) and testing ( using test orders)

**Misc Features**

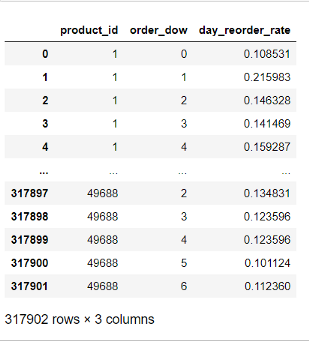
**a. Product features based on time**



order\_hour\_of\_day features

feature : reorder frequency of a product given any hour of the day

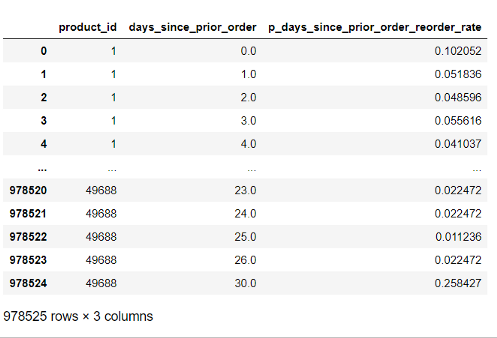
**b. Product features based on day of week**



day\_reorder\_rate

feature: What is the reorder frequency of any product given any day of week ?

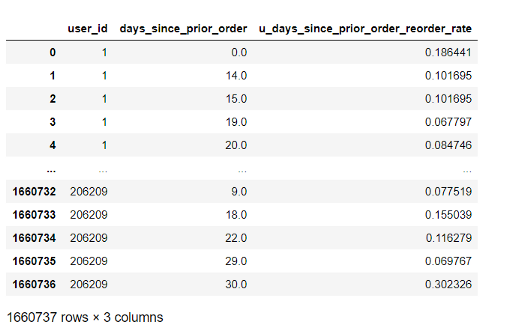
**c. Product features based on difference between 2 orders**



p\_days\_since\_prior\_order\_reorder\_rate

feature: how frequently a product was reordered given a difference between 2 orders (days) contains the product.

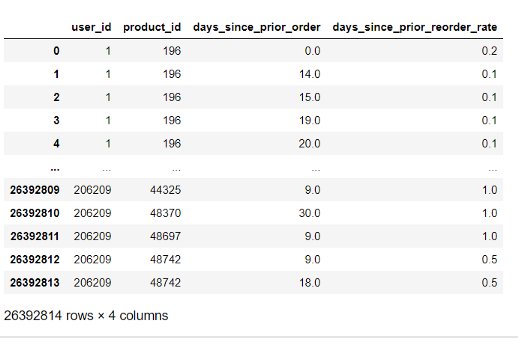
**d. User feature based on difference between 2 orders**



u\_days\_since\_prior\_order\_reorder\_rate

feature: how frequently user reorders any product given a difference between 2 order (days).

**e. User — product reorder rate based on difference between 2 orders**



days\_since\_prior\_reorder\_rate

feature: How frequently user reordered a product given difference between 2 orders (days).

**Merge all data (Best thing about pandas)**

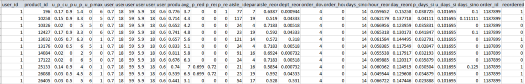
Image ref : <https://iamluminousmen-media.s3.amazonaws.com/media/introduction-to-pyspark-join-types/introduction-to-pyspark-join-types-7.jpg>

# ****Step 7: Generate Training Data and Test data****

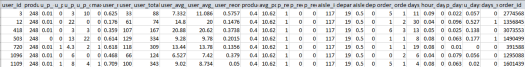
This code will merge the merged\_df (from above ) with the train orders data,

similarly , we will merge test data with merged\_df(from above) to generate test data.

lets have a sneak peek at our training data and test data



training data



test data

As we can see , we do not have reordered column for test data ( we will predict that).

**Step 8: Additional step**

before we start training models, we will reduce the size of our dataframe (currently 3 GB) to 0.6 GB, by changing the default dtypes of columns . Changing dtype of columns from their default value ie. int64/float64/object to their lower range reduced the size by almost 3 times.

Ex — int64 → uint8 ( for department ID , Aisle ID)

We save this file to HDF5 format as saving to CSV after changing the dtypes , resets the dtypes back to default.

# Step 9: Training Models

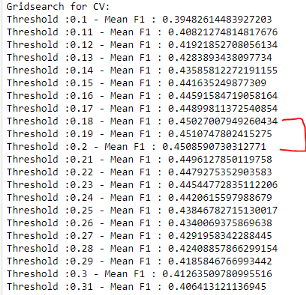
This will be comparison study based on different approaches, and selecting the best performing one. For each models, we decide performance based on score on kaggle and logloss.

We will use 2 approaches to get results,

**a. Global Threshold (0.18 , 0.19, 0.2 ):**

These global thresholds are selected based on:

* **Adhoc approach**: Uploaded many submission files using different thresholds, and saw that after 0.2 F1- score started to decrease.
* **Using strategy 1, discussed above**: we tested on train\_orders, to get an global thresholds



Global thresholds

As seen above Mean F1 — drops after probability threshold 0.2. and highest were 0.18, 0.19, 0.2

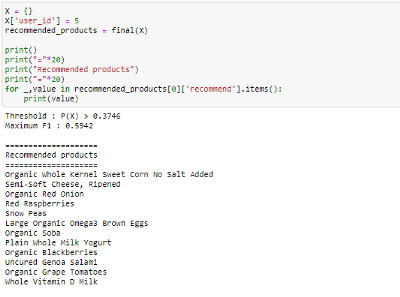
For every model, we will generate 3 results ( submission files) for above thresholds.

We will pick only those products whose predicted reorder probability ≥ given threshold else None.

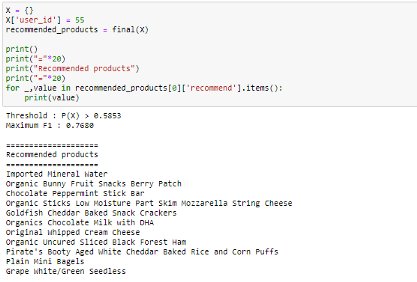
**b. Local Threshold ( F1 - Maximization)**

As described in last post , we will use Faron’s implementation of F1- Maximization such that every order will have its own local threshold and pick those products which will maximize the F1- score.

Here are some examples , where there can be different thresholds for different orders. ( examples shown here are generated to debug the model after training them)



Recommendations for User ID : 5

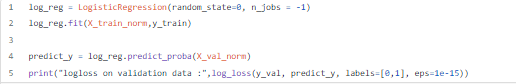


Recommendations for User ID : 55

As seen from above examples , local threshold for every order can boost the F1 Score.

Let’s train some models,

**Model 1 : Logistic Regression**



**Logloss on validation data : 0.2550918280106341**

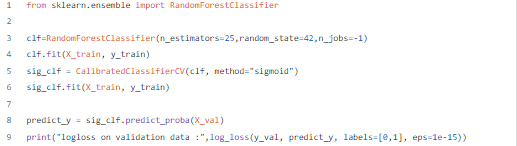
**Model 2: Decision Tree**

**Code:**

|  |
| --- |
| param\_grid={} |
|  | param\_grid['max\_depth'] = [5,10,15,20] |
|  | param\_grid['min\_samples\_split'] = [2,3,4,5] |
|  |  |
|  | dt\_clf = DecisionTreeClassifier() |
|  | r\_search = RandomizedSearchCV(dt\_clf, param\_distributions=param\_grid, cv = 5, verbose = True, n\_jobs = -1) |
|  | r\_search.fit(X\_train, y\_train) |
|  |  |
|  | predict\_y = r\_search.predict\_proba(X\_val) |
|  | print("logloss on validation data :",log\_loss(y\_val, predict\_y, labels=[0,1], eps=1e-15)) |

**Logloss on validation data : 0.2509911734828939**

**Model 3: Random Forest Classifier**



**Logloss on validation data : 0.25187675305313206**

**Model 4: Multi - Layer Perceptron Model**

MLP model

Code:

|  |
| --- |
| Def mlp\_model(): |
|  | """ |
|  | Create mlp model with 4 hidden layers . |
|  | """ |
|  | inp = Input(shape = (28,)) #no of features |
|  | mlp = Dense(256, activation = "relu", kernel\_initializer=tf.keras.initializers.he\_normal(seed=SEED\_VALUE), name = 'fc\_1')(inp) |
|  | mlp = Dropout(0.3)(mlp) |
|  | mlp = Dense(128, activation = 'relu', kernel\_initializer=tf.keras.initializers.he\_normal(seed=SEED\_VALUE), name = 'fc\_2')(mlp) |
|  | mlp = Dropout(0.3)(mlp) |
|  | mlp = Dense(64, activation = 'relu', kernel\_initializer=tf.keras.initializers.he\_normal(seed=SEED\_VALUE), name = 'fc\_3')(mlp) |
|  | mlp = Dropout(0.3)(mlp) |
|  | mlp = Dense(32, activation = 'relu', kernel\_initializer=tf.keras.initializers.he\_normal(seed=SEED\_VALUE), name = 'fc\_4')(mlp) |
|  | mlp = Dropout(0.3)(mlp) |
|  | mlp = Dense(1, activation = 'sigmoid', kernel\_initializer=tf.keras.initializers.he\_normal(seed=SEED\_VALUE), name = 'fc\_5')(mlp) |
|  |  |
|  | model = Model(inputs = inp, outputs = mlp) |
|  |  |
|  | opt = Adam(lr= 0.001) |
|  | model.compile(loss="binary\_crossentropy", optimizer=opt, metrics=["accuracy"]) |
|  |  |
|  | return model |
|  |  |
|  | #train model |
|  | batch\_size = 128 |
|  | history = model.fit(X\_train\_norm,np.array(y\_train).reshape(-1,1), epochs = 10, validation\_data = (X\_val\_norm,np.array(y\_val).reshape(-1,1)), callbacks=[checkpoint, earlystop]) |
|  |  |
|  | #load the saved model and print accuracy |
|  |  |
|  | saved\_model = tf.keras.models.load\_model('saved\_model/mlp/checkpoint.hdf5') |
|  | print("Training Accuracy :{:.2f} %".format(saved\_model.evaluate(X\_train\_norm,np.array(y\_train).reshape(-1,1), verbose = 0)[1]\*100)) |
|  | print("Validation Accuracy :{:.2f} %".format(saved\_model.evaluate(X\_val\_norm,np.array(y\_val).reshape(-1,1), verbose = 0)[1]\*100)) |

**Training Accuracy : 90.75 %  
Validation Accuracy : 90.74 %  
Logloss on validation data : 0.2513314122715033**

**Model 5 : XGBoost Classifier**

**Code:**

|  |
| --- |
| def train\_xgb(X\_train, X\_test, y\_train, y\_test, plot\_importance = True, save = True, file\_name = None, params = None): |
|  |  |
|  | """ |
|  | Returns trained XGB model and output probabilities of validation set |
|  |  |
|  | Parameters |
|  | ---------- |
|  | X\_train : X\_train data is passed |
|  | X\_test : X\_test data is passed |
|  | y\_train : y\_train data is passed |
|  | y\_test : y\_test data is passed |
|  | plot\_importance : The default is True |
|  | Boolean variable , Set to True if feature Importance |
|  | needs to be plotted |
|  | save : The default is True |
|  | Boolean variable , Set to True if trained model |
|  | needs to be saved |
|  | file\_name : The default is None |
|  | Filename to be used while saving model |
|  | params : The default is None |
|  | dict of parameter set used to train XGBoost model |
|  | Returns |
|  | ------- |
|  | xgb\_model : Trained XGBoost Model |
|  | predict\_y : output probabilities for validation data |
|  | """ |
|  |  |
|  | #defining set of parameters, these are optimal parameters obatined after |
|  | # rigorous hyperparameter Tuning |
|  | if params is None: |
|  | params = {} |
|  | params['objective'] = 'binary:logistic' |
|  | params['eval\_metric'] = ['logloss'] |
|  | params['eta'] = 0.02 |
|  | params['max\_depth'] = 15 |
|  | params['nthread']=-1 |
|  | params['colsample\_bytree'] = 0.4 |
|  | params['tree\_method'] = 'gpu\_hist' |
|  |  |
|  | #Create DataMatrix for XGBoost |
|  | d\_train = xgb.DMatrix(X\_train, label=y\_train) |
|  | d\_test = xgb.DMatrix(X\_val, label=y\_val) |
|  |  |
|  | watchlist = [(d\_train, 'train'), (d\_test, 'valid')] |
|  |  |
|  | #output Training Time |
|  | start\_time = datetime.now() |
|  | print("Training Started :") |
|  | xgb\_model = xgb.train(params, d\_train, 300, watchlist, early\_stopping\_rounds=20, verbose\_eval=10) |
|  | print("Training Completed ") |
|  | end\_time = datetime.now() |
|  | difference = end\_time - start\_time |
|  | time = divmod(difference.total\_seconds() , 360) |
|  | print("Total Time : {} minutes {} seconds".format(time[0], time[1])) |
|  |  |
|  | #get probabilities for validation set |
|  | predict\_y = xgb\_model.predict(d\_test) |
|  | print("The test log loss is:",log\_loss(y\_val, predict\_y, labels=[0,1], eps=1e-15)) |
|  |  |
|  | #save trained model |
|  | if save: |
|  | #file\_name = "xgb\_v1.pkl" |
|  |  |
|  | # save |
|  | pickle.dump(bst, open(file\_name, "wb")) |
|  |  |
|  | #plot Feature Importance |
|  | if plot\_importance: |
|  | print("Feature Importance") |
|  | fig, ax = plt.subplots(1,1,figsize=(10,10)) |
|  | xgb.plot\_importance(booster = bst, ax = ax) |
|  |  |
|  | return xgb\_model, predict\_y |

**Logloss on validation data : 0.24345293402046597**

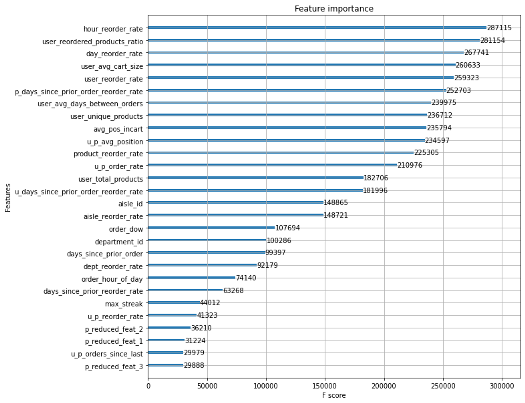


Fig: feature Importance using XGBoost

We can see that hour\_reorder\_rate ( one of misc features) has highest importance.

**Model 6 : CatBoost Classifier**

**Code:**

|  |
| --- |
| def train\_catboost(X\_train, X\_test, y\_train, y\_test, plot\_importance = True, save = True, file\_name = None): |
|  |  |
|  | """ |
|  | Returns trained catboost model and output probabilities of validation set |
|  |  |
|  | Parameters |
|  | ---------- |
|  | X\_train : X\_train data is passed |
|  | X\_test : X\_test data is passed |
|  | y\_train : y\_train data is passed |
|  | y\_test : y\_test data is passed |
|  | plot\_importance : The default is True |
|  | Boolean variable , Set to True if feature Importance |
|  | needs to be plotted |
|  | save : The default is True |
|  | Boolean variable , Set to True if trained model |
|  | needs to be saved |
|  | file\_name : The default is None |
|  | Filename to be used while saving model |
|  | Returns |
|  | ------- |
|  | xgb\_model : Trained XGBoost Model |
|  | predict\_y : output probabilities for validation data |
|  | """ |
|  |  |
|  |  |
|  | start\_time = datetime.now() |
|  | print("Training Started :") |
|  |  |
|  | #defining set of parameters, these are optimal parameters obatined after |
|  | # rigorous hyperparameter Tuning |
|  | c\_model = CatBoostClassifier(task\_type = "GPU",verbose=True,depth = 13, iterations= 2000,learning\_rate= 0.02,scale\_pos\_weight= 1.0) |
|  | c\_model.fit(X\_train,y\_train) |
|  | print("Training Completed ") |
|  | end\_time = datetime.now() |
|  | difference = end\_time - start\_time |
|  | time = divmod(difference.total\_seconds() , 3600) |
|  | print("Total Time : {} hours {} seconds".format(time[0], time[1])) |
|  |  |
|  | #get output probabilities |
|  | predict\_y = c\_model.predict\_proba(X\_test) |
|  | print("The Test log loss is:",log\_loss(y\_test, predict\_y, labels=[0,1], eps=1e-15)) |
|  |  |
|  | #get probs for class 1 |
|  | predict\_y = predict\_y[:,-1] |
|  |  |
|  | #save the model |
|  | if save: |
|  | # save |
|  | pickle.dump(c\_model, open(file\_name, "wb")) |
|  |  |
|  | #plot feature importance |
|  | if plot\_importance: |
|  | #ref: https://stackoverflow.com/a/65842279/11533069 |
|  | f\_imp=pd.DataFrame({'features':train\_x.columns.to\_numpy(),'feature\_importance': c\_model.get\_feature\_importance()}) |
|  | f\_imp.sort\_values(by = 'feature\_importance', ascending = False, inplace = True) |
|  |  |
|  | print("Feature Importance") |
|  | fig, ax = plt.subplots(1,1,figsize=(10,10)) |
|  | sns.barplot(x=f\_imp['feature\_importance'], y=f\_imp['features']) |
|  | plt.title('Feature Importance') |
|  | plt.xlabel('Importance') |
|  | plt.ylabel('Features') |
|  |  |
|  | return c\_model, predict\_y |

**Logloss on validation data : 0.24300858358388394**

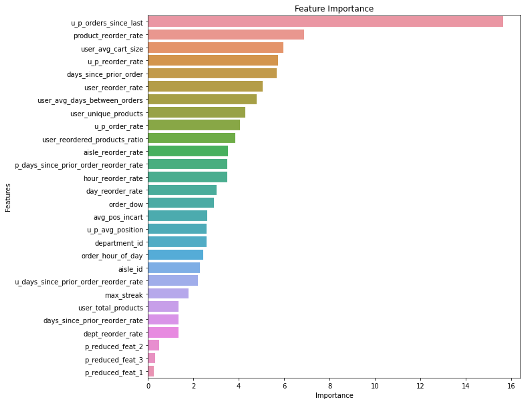


Fig: feature importance using catboost

We can see that u\_p\_orders\_since\_last has highest importance, which in contrast to xgb was in 2nd last.

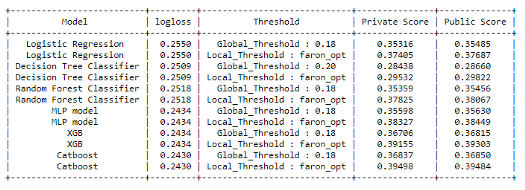
Note: **days\_since\_prior\_order** is an important feature here for both XGB and Catboost model. But here is the catch, for any future order after deploying the model, we cant have **days\_since\_prior\_order**value, as we don’t know user’s last order **date.**We will handle this in our deployment section.

# ****Generate Submission Files****

lets generate submission files, (for both global and local thresholds)

* The function globl\_threshold\_products → generates submission files based on global thresholds (0.18, 0.19, 0.20)
* The function getscores\_on\_testdata → generates submission file based on local threshold to maximize F1- Score

**Step 10: Model comparison based on performance**



Model performance

We see catboost classifier on local threshold scored highest

# ****Step 11: Improvising the model****

We can improve the Catboost Model slightly

During my trials on different models , I found out that , instead of splitting our training data randomly into training and validation set for training our models,**if we split training data on users , we can improve our models.**

Although this step improves model slightly , it was significant enough to mention here.

Code:

|  |
| --- |
| def random\_split(test\_size = 0.1): |
|  |  |
|  | """ |
|  | Split data randomly into train and validation data |
|  | """ |
|  | train\_y = train\_data['reordered'].values.tolist() |
|  | train\_x = train\_data.drop(['user\_id', 'product\_id', 'order\_id', 'reordered'], axis = 1) |
|  |  |
|  | # since there are Millions of data ,we are taking 10% of data in test set , |
|  | X\_train, X\_val, y\_train, y\_val = train\_test\_split(train\_x, train\_y, stratify=train\_y, test\_size=0.1, random\_state = 42) |
|  |  |
|  | #display distribution of data |
|  | display\_target\_distribution(y\_train, y\_val) |
|  | return (X\_train, y\_train),(X\_val, y\_val) |
|  |  |
|  | def split\_by\_user(test\_size = 0.1): |
|  |  |
|  | #split on users |
|  | sampled\_users = train\_data['user\_id'].sample(n=int(0.1 \* train\_data['user\_id'].nunique()), random\_state=42,replace=False).reset\_index() |
|  | val = train\_data[train\_data['user\_id'].isin(sampled\_users['user\_id'].tolist())] |
|  | train = train\_data[~train\_data['user\_id'].isin(sampled\_users['user\_id'].tolist())] |
|  |  |
|  | y\_train = train['reordered'].values.tolist() |
|  | X\_train = train.drop(['user\_id', 'product\_id', 'order\_id', 'reordered'], axis = 1) |
|  |  |
|  | y\_val = val['reordered'].values.tolist() |
|  | X\_val = val.drop(['user\_id', 'product\_id', 'order\_id', 'reordered'], axis = 1) |
|  |  |
|  | display\_target\_distribution(y\_train, y\_val) |
|  | return (X\_train, y\_train),(X\_val, y\_val) |

That marks the end of our modelling stage.

Step 12: Cold start Problems:

1. **New user : What products can we recommend to a new user ?**

There can be many solutions for this , few of them can be :

* provide most frequently purchased product
* provide most frequently purchased products based on hour and day of week. ( we will use this one)

2. **days\_since\_prior\_order : What to do when we don’t have user’s last order date ?**

As mentioned in the previous post, **days\_since\_prior\_order**is an important feature, but after deploying the model, for a future order we can calculate this value only if we have user’s last order date.

To handle this, I have assumed that all users placed their last order on 21–3–2021 (assuming that is the time of deployment ) , and for any future order , I can calculate the difference in days , thus giving us **days\_since\_prior\_order**feature.

Note:

This change introduced new challenges, such as there will be users , who never made a purchase in a window of ’n’ days where n ={0,1,…}. and we don’t have some **misc. features** ( specifically p\_days\_since\_prior\_order\_reorder\_rate, u\_days\_since\_prior\_order\_reorder\_rate and days\_since\_prior\_reorder\_rate , refer last post) as those combination might not exist in our training set.

For such cases, features depending on days\_since\_prior\_order in misc. features are set to 0.0

# Solutions for Cold Start Problems

**Prediction for new users**

We will now generate a pickle file with top 10 products for each hour for any given day of week



Top 10 products of 5th hour of 5th day of week

This will be used in cold start problem of New User

**User last order date**

Now that we have established the importance of days\_since\_prior\_order feature , so we need to generate a file containing last order date of every user. Which in this case is 21–3–2021.



user\_last\_purchase

# Build the Pipeline

* Get user id as Input
* Get current time and day of week using python datetime package.

Get current time and day of week

We will use python datetime package to extract **Order\_hour\_of\_day** and **Order\_dow** from current time

* Read user\_last\_purchase.pkl
* If user is not in user\_last\_purchase.pkl, i.e. a new user, then read top10\_products.pkl

New user predictions

If the user doesn’t exist in user\_last\_purchase.pkl, we will select 10 products from top10\_products.pkl based on **Order\_hour\_of\_day** and **Order\_dow.**

* If it is an existing user, calculate days\_since\_prior\_order

https://miro.medium.com/max/525/1*XEUEELe7HttBnGGfcYUXdg.png

days\_since\_prior\_order

user\_last\_order\_date = ulp[ulp['user\_id']==user\_id]['date'].values.tolist()[0]days\_since\_prior\_order = today - int(user\_last\_order\_date.split('-')[-1])  
   
del ulp, now, today, dt\_string, user\_last\_order\_date

* Read all files which were built while generating features

read all necessary files

Filter these files based on user ID, order\_dow, order\_hour\_of\_day and days\_since\_prior\_order.

Note:

Since there are 206209 users in total and entire training set was of shape (8474661 , 32). So, we saved intermediate files, which we will filter based on user ID and merge them to generate features at the run time.

* Handle features based on days\_since\_prior\_order feature

As discussed in cold start section, there might be cases when, we don’t have some **misc. features** ( features based on days\_since\_prior\_order) as those combination might not exist in our training set. We will generate a new rows and set those values to 0.0

* Inner join based on user ID
* User these features to predict the reorder probability

First we load the model, then predict using the model. Finally , use F1- Maximization to get most probable products for this user

This entire pipeline is wrapped inside a function get\_recommendations()

# Step 13: Build Web-API



Image [Source](https://i.ytimg.com/vi/Hti3LCocY8k/maxresdefault.jpg)

A web API allows for information to be manipulated by other programs over the network. Flask is a web framework for Python, which helps in building web applications.

We build simple a Web-API now.

**STEP 14: Code Walkthrough**

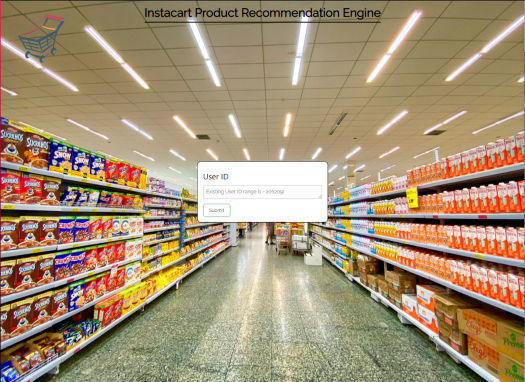
* Import all necessary libraries
* Import get\_recommendations from get\_predictions.py → This will give recommendations based on user ID
* Create a Flask object with name app with variable \_\_name\_\_ , which will be accessed by \_\_main\_\_ .
* If the homepage is accessed i.e. URL with (‘/’), then the decorator @app.route(’/’) will execute home() function and flask with render index.html → a html file for homepage. By default, a routeonly answers to GET requests.
* Once user enters the User ID ( new / existing), we will accept this request as a dictionary in our predict function.
* predict function generates recommendations using get\_recommendation function which is explained above.
* From this point we will redirect to different webpages if user is new or existing.
* Since predict is decorated with @app.route(’/predict’, method = ['POST]) , we will redirect to **new\_user\_recommendation.html** if its a new user or to **predict.html** if its the existing user and post the data in form of dictionary.
* These webpages will display the recommendations.

I deployed this application locally at 0.0.0.0/8000

# Step 15:Predictions

local deployment of the model

* **The landing page**



homepage

* **For existing user, say User\_ID = 206209**



Recommendation for Existing User

The recommendations are based on user’s previous purchases

* **For a new user, say User\_ID = 226172**



Recommendation for New user

These recommendations are based on most frequently bought items at this hour, on this day of week.