WeCloudData

# Instacart Market Basket Analysis

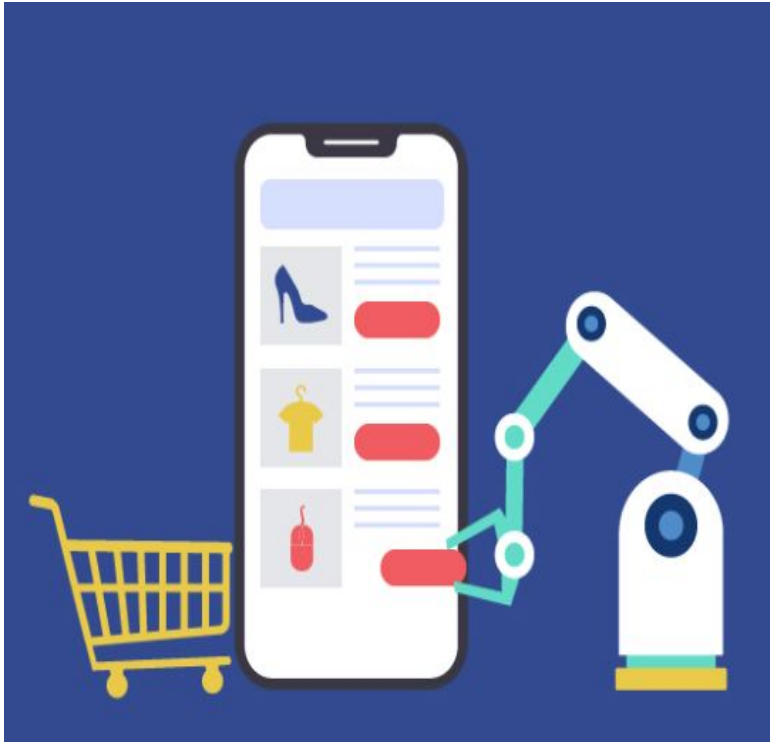
Allen XU

# Agenda

- Introduction
- Business Goal
- Exploratory Data Analysis
- Feature Engineering
- Model Select
- Model Interpretation
- Apply Model
- Challenge & Next Step

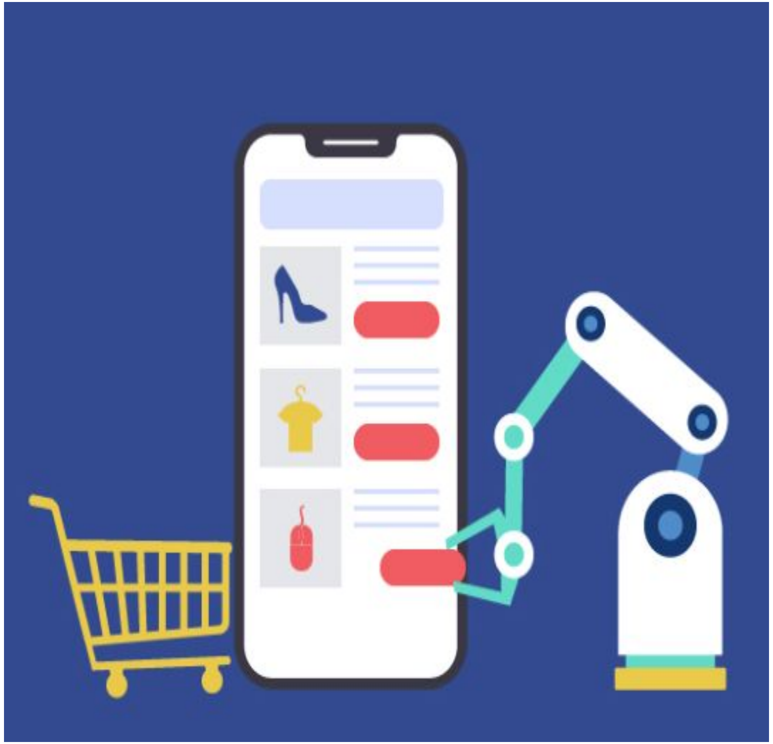
# Intro ML For Retail

- Personalized Marketing
- Demand Forcasting
- Supply Chain
- Custermer Experience



# Personalized Marketing

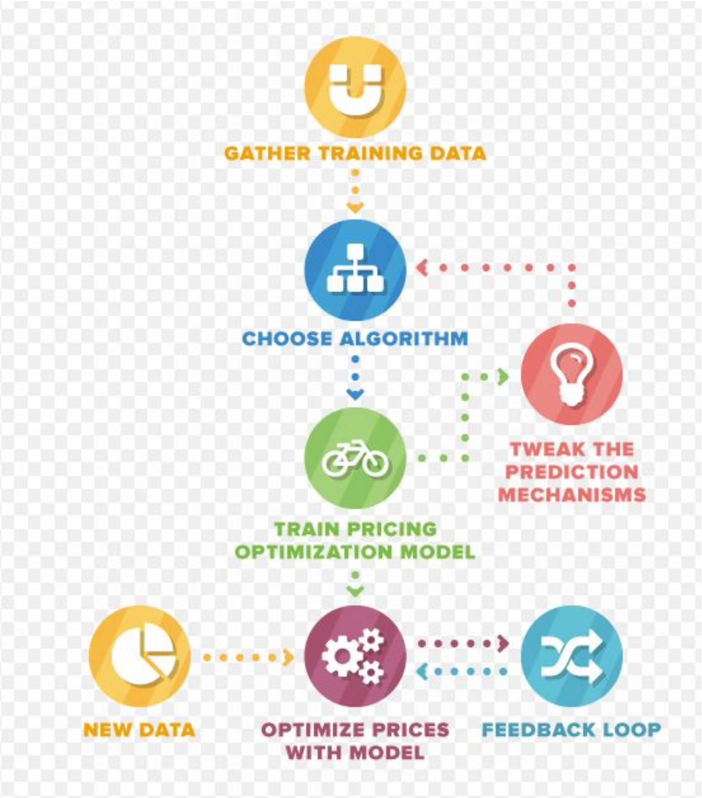
- Others You May Like
- Frequently Bought Together (shopping cart expansion)
- Recommended for You
- Similar Items
- **Buy it Again** ⭐
- On-sale / Deals
- Recently Viewed



https://cloud.google.com/retail/docs/models#introduction

# Goal

- Set Up Business Goal
  - Simplify user purchase journey (short term)
  - improve user retention rate (long term)
- Define Machine Learning Problem
  - Binary classification: Will Reorder/Won't Reorder
  - Statistic Metric
    - F1
    - Accuracy



# Data

```
order_products_train_df.head()
```

| | order_id | product_id | add_to_cart_order | reordered |
|---|---|---|---|---|
| **0** | 1 | 49302 | 1 | 1 |
| **1** | 1 | 11109 | 2 | 1 |
| **2** | 1 | 10246 | 3 | 0 |
| **3** | 1 | 49683 | 4 | 0 |
| **4** | 1 | 43633 | 5 | 1 |

```
order_products_prior_df.head()
```

| | order_id | product_id | add_to_cart_order | reordered |
|---|---|---|---|---|
| **0** | 2 | 33120 | 1 | 1 |
| **1** | 2 | 28985 | 2 | 1 |
| **2** | 2 | 9327 | 3 | 0 |
| **3** | 2 | 45918 | 4 | 1 |
| **4** | 2 | 30035 | 5 | 0 |

```
aisles_df.head()
```

| | aisle_id | aisle |
|---|---|---|
| **0** | 1 | prepared soups salads |
| **1** | 2 | specialty cheeses |
| **2** | 3 | energy granola bars |
| **3** | 4 | instant foods |
| **4** | 5 | marinades meat preparation |

```
aisles_df.shape
```

```
(134, 2)
```

```
departments_df.head()
```

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

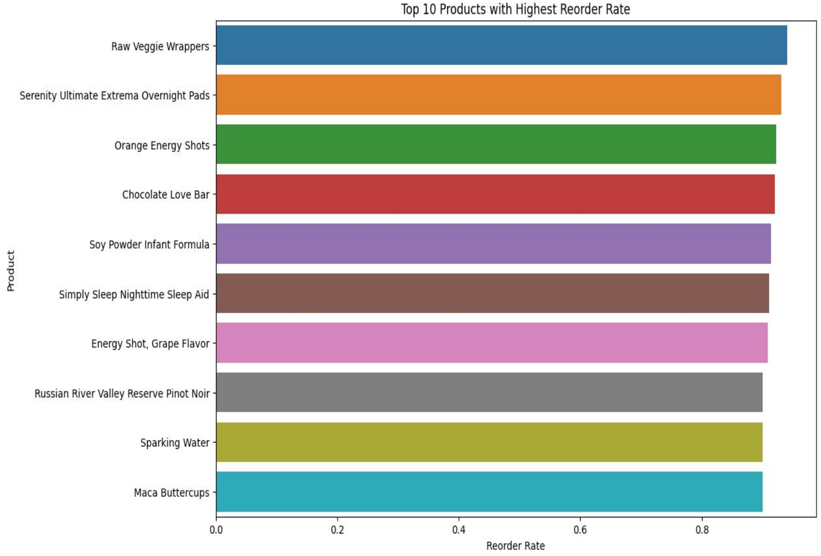
```
orders_df.head()
```

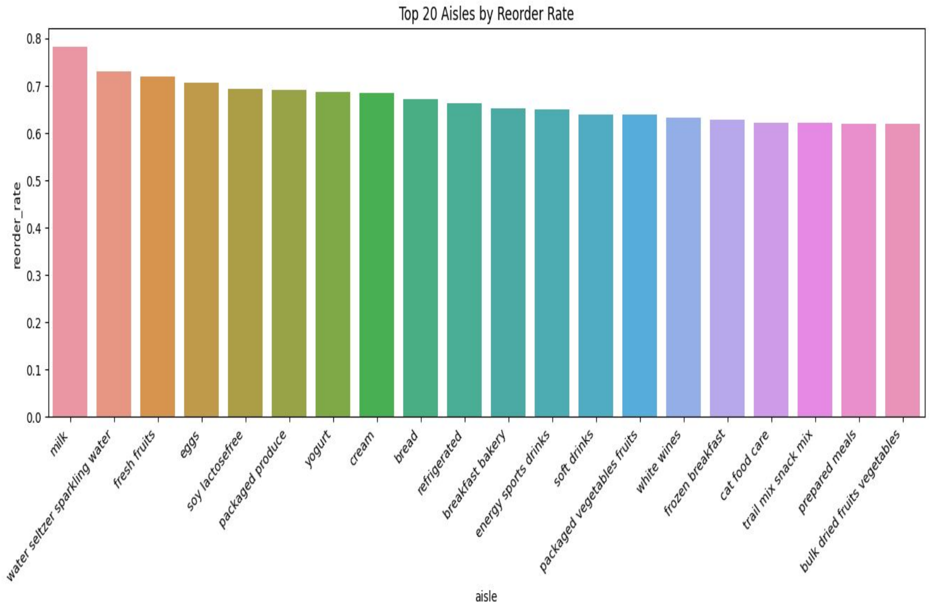
| | order_id | user_id | eval_set | order_number | order_dow | order_hour_of_day | days_since_prior_order |
|---|---|---|---|---|---|---|---|
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN |
| **1** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 |
| **2** | 473747 | 1 | prior | 3 | 3 | 12 | 21.0 |
| **3** | 2254736 | 1 | prior | 4 | 4 | 7 | 29.0 |
| **4** | 431534 | 1 | prior | 5 | 4 | 15 | 28.0 |

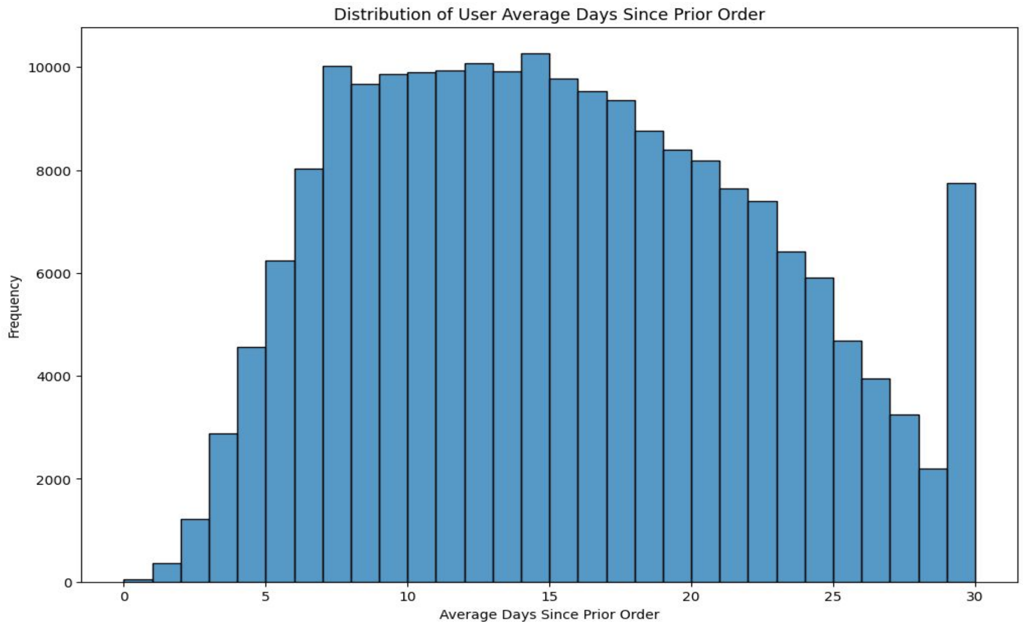
```
products_df.head()
```

| | product_id | product_name | aisle_id | department_id |
|---|---|---|---|---|
| **0** | 1 | Chocolate Sandwich Cookies | 61 | 19 |
| **1** | 2 | All-Seasons Salt | 104 | 13 |
| **2** | 3 | Robust Golden Unsweetened Oolong Tea | 94 | 7 |
| **3** | 4 | Smart Ones Classic Favorites Mini Rigatoni Wit... | 38 | 1 |
| **4** | 5 | Green Chile Anytime Sauce | 5 | 13 |

# Data Exploration




Distribution of User Average Days Since Prior Order
Frequency
0
2000
4000
6000
8000
10000
0
5
10
15
20
25
30
Average Days Since Prior Order

# Feature Engineering

## User Part

```python
# _user_total_orders:
# _user_sum_days_since_prior_order:
# _user_mean_days_since_prior_order:
agg_dict_2 = {'order_number': [('_user_total_orders', 'max')],
              'days_since_prior_order': [('_user_sum_days_since_prior_order', 'sum'),
                                         ('_user_mean_days_since_prior_order', 'mean')]}

users = ka_add_groupby_features_1_vs_n(orders[orders.eval_set == 'prior'], ['user_id'], agg_dict_2)

# _user_reorder_ratio: total number of reorder/total number of orders of first order
# _user_total_products:
# _user_distinct_products:
agg_dict_3 = {'reordered': [('_user_reorder_ratio',
                             lambda x: sum(priors_orders_detail.loc[x.index, 'reordered'] == 1) /
                                       sum(priors_orders_detail.loc[x.index, 'order_number'] > 1))],
              'product_id': [('_user_total_products', 'count'),
                             ('_user_distinct_products', lambda x: x.nunique())]}

us = ka_add_groupby_features_1_vs_n(priors_orders_detail, ['user_id'], agg_dict_3)
users = users.merge(us, how='inner')

# average number of item in each orders
users['_user_average_basket'] = users['_user_total_products'] / users['_user_total_orders']

us = orders[orders.eval_set != "prior"][['user_id', 'order_id', 'eval_set', 'days_since_prior_order']]
us.rename(columns={'days_since_prior_order': 'time_since_last_order'}, inplace=True)

users = users.merge(us, how='inner')
```

## Item part

```python
# add order information to priors set
priors_orders_detail = orders.merge(right=priors, how='inner', on='order_id')

# create new variables
# _user_buy_product_times: the time user by this item
priors_orders_detail['_user_buy_product_times'] = priors_orders_detail.groupby(['user_id', 'product_id']).cumcount() + 1

# _prod_tot_cnts: total number that item been sold
# _reorder_tot_cnts_of_this_prod: reorder number of this item
# _prod_order_once: the number of times that item only buy 1
# _prod_order_more_than_once: the number of times that item buy more that once
agg_dict = {'order_id': [('_total_orders', 'count')],
            'add_to_cart_order': [('_sum_add_to_cart_order', 'sum')],
            'reordered': [('_reorder_total_cnt', 'sum')],
            '_user_buy_product_times': [('_prod_buy_first_time_total_cnt', lambda x: sum(x == 1)),
                                        ('_prod_buy_second_time_total_cnt', lambda x: sum(x == 2))]}

prd = ka_add_groupby_features_1_vs_n(priors_orders_detail, ['product_id'], agg_dict)

# _prod_reorder_prob:
# _prod_reorder_ratio:
prd['_prod_reorder_prob'] = prd['_prod_buy_second_time_total_cnt'] / prd['_prod_buy_first_time_total_cnt']
prd['_prod_reorder_ratio'] = prd['_reorder_total_cnt'] / prd['_total_orders']
prd['_prod_reorder_times'] = 1 + prd['_reorder_total_cnt'] / prd['_prod_buy_first_time_total_cnt']
```

## General Part

```python
# _up_order_count: number of times user buy this item
# _up_first_order_number: order number of user that first time buy this item
# _up_last_order_number: order number of user that last time buy this item
# _up_average_cart_position:
agg_dict_4 = {'order_number': [('_up_order_count', 'count'),
                               ('_up_first_order_number', 'min'),
                               ('_up_last_order_number', 'max')],
              'add_to_cart_order': [('_up_average_cart_position', 'mean')]}

data = ka_add_groupby_features_1_vs_n(df=priors_orders_detail,
                                      group_columns_list=['user_id', 'product_id'],
                                      agg_dict=agg_dict_4)

data = data.merge(prd, how='inner', on='product_id').merge(users, how='inner', on='user_id')
# _up_order_rate :number of times buy this item/ total order number
# _up_order_since_last_order
# _up_order_rate_since_first_order : number of purchase of this item/ the number of orders between first time buy this item and last time buy this item 该
data['_up_order_rate'] = data['_up_order_count'] / data['_user_total_orders']
data['_up_order_since_last_order'] = data['_user_total_orders'] - data['_up_last_order_number']
data['_up_order_rate_since_first_order'] = data['_up_order_count'] / (data['_user_total_orders'] - data['_up_first_order_number'] + 1)

# add user_id to train set
train = train.merge(right=orders[['order_id', 'user_id']], how='left', on='order_id')
data = data.merge(train[['user_id', 'product_id', 'reordered']], on=['user_id', 'product_id'], how='left')

# release Memory
del priors_orders_detail, orders
gc.collect()
data.head()
```

# Model Select (Metrics F1&Accuracy)

## LR

```python
# Initialize a Logistic Regression model with specifie
lr_model = LogisticRegression(n_jobs=16,
                              random_state=42,
                              class_weight='balanced')

# Fit the model using the training data
lr_model.fit(X_train, y_train)
```

```
F1 Score: 0.2246
Accuracy Score: 0.7964
```

## RF

```python
# Create a Random Forest model instance
rf_model = RandomForestClassifier(n_jobs=16,
                                  random_state=42)

# Fit the model using the training data
rf_model.fit(X_train, y_train)
```

```
F1 Score: 0.2693
Accuracy Score: 0.9085
```

## XGB

```python
xgb_params = {
    "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
    ,"lambda"            :10
    ,"n_jobs"            : 16
}

watchlist = [(d_train, "train")]
bst = xgboost.train(params=xgb_params,
                    dtrain=d_train,
                    num_boost_round=80,
                    evals=watchlist, verbose_eval=10)
```

```
F1 Score: 0.4419
Accuracy Score: 0.8747
```

## DT

```python
# Train a Decision Tree classifier with specified parameters
dt = DecisionTreeClassifier(max_depth=10,
                            min_samples_split=10,
                            min_samples_leaf=10,
                            class_weight='balanced')
dt.fit(X_train, y_train)
```

```
F1 Score: 0.3647
Accuracy Score: 0.7481
```

# Model Select(Kaggle Results)

## XGB

| | order_id | products |
|---|---|---|
| 0 | 17 | 13107 21463 |
| 1 | 34 | 21137 39180 47792 43504 13176 47766 44663 1608... |
| 2 | 137 | 38689 5134 41787 23794 2326 25890 24852 |
| 3 | 182 | 32109 41149 33000 47672 39275 9337 11520 47209... |
| 4 | 257 | 21137 27966 28476 49235 29837 37646 27104 4605... |
| ... | ... | ... |
| 74995 | 3420740 | 19660 13176 16291 49005 8174 35951 39146 46252 |
| 74996 | 3420877 | 21137 27966 43122 10768 13176 49111 12238 8277... |
| 74997 | 3420888 | 20542 43961 44632 22935 35951 46906 28985 4776... |
| 74998 | 3420989 | 5134 46676 47229 21616 42450 35004 43210 47766... |
| 74999 | 3421054 | 18426 38448 11123 31231 13375 24852 |

75000 rows × 2 columns

| | product_id | product_name |
|---|---|---|
| 13106 | 13107 | The Complete Cookie Double Chocolate |
| 21462 | 21463 | Vanilla Bean Light Ice Cream |

## DT

| | order_id | products |
|---|---|---|
| 0 | 17 | 31964 13107 21903 39275 39928 21463 38777 |
| 1 | 34 | 21137 7948 13176 30353 49191 2596 39180 47792 ... |
| 2 | 137 | 38689 24235 43352 29594 23794 25890 34126 5134... |
| 3 | 182 | 32109 28800 22935 15252 47672 13629 40377 3927... |
| 4 | 257 | 21137 38693 24838 30233 24964 27966 29837 2203... |
| ... | ... | ... |
| 74995 | 3420740 | 19660 11777 38772 46252 5785 16291 39928 10017... |
| 74996 | 3420877 | 21137 11005 5449 6567 5918 7781 13176 12238 26... |
| 74997 | 3420888 | 31506 22935 28985 42701 24964 30450 42123 4396... |
| 74998 | 3420989 | 45866 7781 42450 28985 30406 27521 5134 21616 ... |
| 74999 | 3421054 | 27429 18426 5818 38448 27370 11123 15732 15802... |

75000 rows × 2 columns

| | product_id | product_name |
|---|---|---|
| 13106 | 13107 | The Complete Cookie Double Chocolate |
| 21462 | 21463 | Vanilla Bean Light Ice Cream |
| 21902 | 21903 | Organic Baby Spinach |
| 31963 | 31964 | Passionfruit Sparkling Water |
| 38776 | 38777 | Organic Green Seedless Grapes |
| 39274 | 39275 | Organic Blueberries |
| 39927 | 39928 | Organic Kiwi |

## LR

| | order_id | products |
|---|---|---|
| 0 | 17 | 16965 21709 15613 21903 39275 39928 47766 3877... |
| 1 | 34 | 21137 13176 39275 47766 46979 42265 44632 |
| 2 | 137 | 26209 24184 18465 41787 21903 43352 16797 2710... |
| 3 | 182 | 31717 17794 21903 13176 27104 44359 39275 2293... |
| 4 | 257 | 21137 13176 24838 28985 30233 42701 24964 2796... |
| ... | ... | ... |
| 74995 | 3420740 | 21137 19660 13176 43352 26604 22935 5077 24964... |
| 74996 | 3420877 | 21137 13176 26604 22935 24964 47626 27966 4605... |
| 74997 | 3420888 | 8193 8424 28204 42736 43961 44632 31506 22935 ... |
| 74998 | 3420989 | 27521 7781 46676 17948 31506 21616 8277 35951 ... |
| 74999 | 3421054 | 28842 24852 |

75000 rows × 2 columns

| | product_id | product_name |
|---|---|---|
| 15612 | 15613 | Salted Tub of Butter |
| 16964 | 16965 | Chocolate Ice Cream |
| 21708 | 21709 | Sparkling Lemon Water |
| 21902 | 21903 | Organic Baby Spinach |
| 38776 | 38777 | Organic Green Seedless Grapes |
| 39274 | 39275 | Organic Blueberries |
| 39927 | 39928 | Organic Kiwi |
| 47765 | 47766 | Organic Avocado |

## RF

| | order_id | products |
|---|---|---|
| 0 | 17 | 13107 |
| 1 | 34 | 16083 |
| 2 | 137 | None |
| 3 | 182 | 9337 13629 39275 |
| 4 | 257 | 49235 |
| ... | ... | ... |
| 74995 | 3420740 | None |
| 74996 | 3420877 | 21137 13176 13646 |
| 74997 | 3420888 | 35951 44632 |
| 74998 | 3420989 | 47766 |
| 74999 | 3421054 | 11123 31231 |

75000 rows × 2 columns

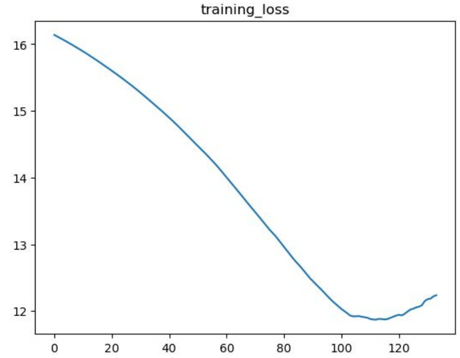
| | product_id | product_name |
|---|---|---|
| 13106 | 13107 | The Complete Cookie Double Chocolate |

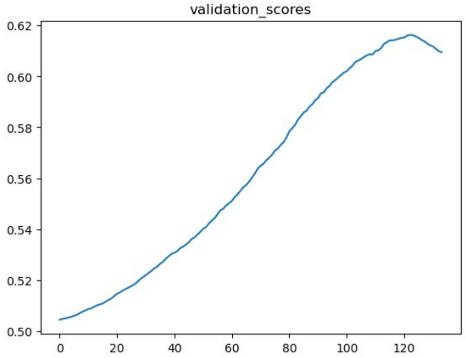
# Deep Learning Model

- MLPClassifier
- ‘sgd’ refers to stochastic gradient descent.
- Adam solver with early_stopping to prevent overfir

```
model = MLPClassifier(solver='adam',
                      activation ='relu',
                      learning_rate_init = 0.0000001,
                      alpha=1e-5,
                      hidden_layer_sizes=(128, 32, 16),
                      random_state=1,
                      warm_start=True,
                      batch_size = 4096,
                      verbose = True,
                      tol=1e-10,
                      early_stopping=True)

model.fit(X_train.values, y_train.values)
```



# Ensemble Methods

```python
xgb_params = {
    "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
    ,"lambda"           :10
    ,"n_jobs"           : 16
}

# Create an XGBoost model
xgb_model = xgboost.XGBClassifier(**xgb_params)

# Create a Decision Tree model
dt_model = DecisionTreeClassifier(max_depth=10, min_samples_split=10, min_samples_leaf=10)

# Create a Logistic Regression model
lr_model = LogisticRegression(n_jobs=16, random_state=42)

# Create a Voting Classifier instance, including all models
voting_classifier = VotingClassifier(estimators=[
    ('xgb', xgb_model),
    ('dt', dt_model),
    ('lr', lr_model)
], voting='hard')

# Fit the model using the training data
voting_classifier.fit(X_train, y_train)

# Make predictions on the test data
X_test.loc[:, 'reordered'] = voting_classifier.predict(X_test.drop(['eval_set', 'user_id', 'order_id', 'reordered', 'product_id'], axis=1))

# Keep the same post-processing steps as before
X_test.loc[:, 'product_id'] = X_test.product_id.astype(str)

submit = ka_add_groupby_features_n_vs_1(X_test[X_test.reordered == 1],
                                        group_columns_list=['order_id'],
                                        target_columns_list=['product_id'],
                                        methods_list=[lambda x: ' '.join(set(x))], keep_only_stats=True)
submit.columns = sample_submission.columns.tolist()

submit_final_hard = sample_submission[['order_id']].merge(submit, how='left').fillna('None')
submit_final_hard
```

## Hard

| | order_id | products |
|---|---|---|
| **0** | 17 | 38777 21903 39928 39275 |
| **1** | 34 | 21137 13176 47766 42265 44632 |
| **2** | 137 | 26209 24184 18465 41787 21903 43352 27104 3412... |
| **3** | 182 | 39275 22935 9337 47209 5479 8518 |
| **4** | 257 | 21137 12341 27966 49235 37646 22035 27104 4605... |
| **...** | ... | ... |
| **74995** | 3420740 | 19660 48679 8174 39928 35951 |
| **74996** | 3420877 | 21137 27966 21903 13176 26604 8277 47209 40706... |
| **74997** | 3420888 | 8193 8424 42736 43961 44632 31506 22935 35951 ... |
| **74998** | 3420989 | 27521 7781 46676 17948 21616 8277 35951 47766 ... |
| **74999** | 3421054 | 11123 31231 24852 |

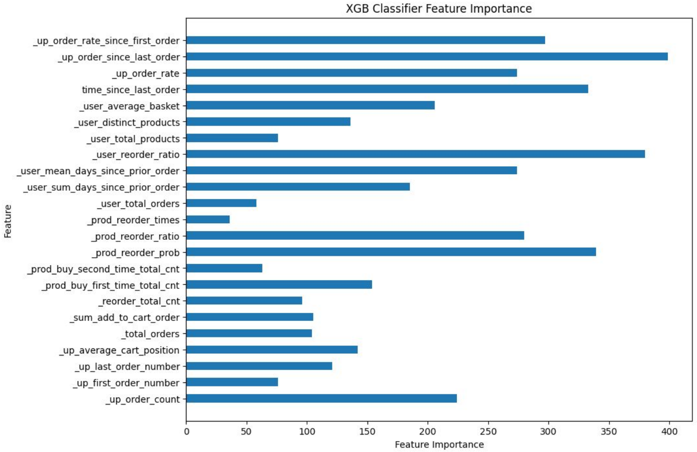
75000 rows × 2 columns

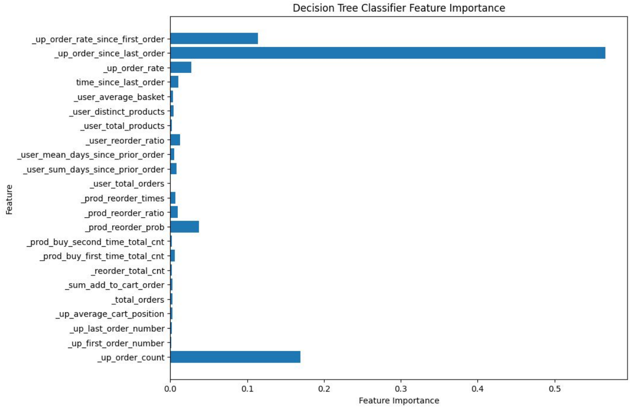
## Soft

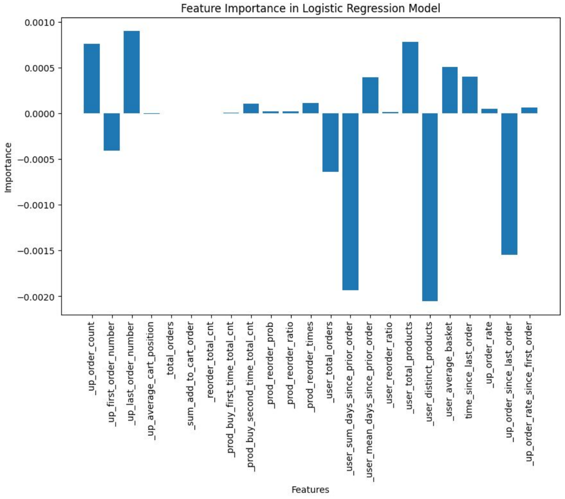
| | order_id | products |
|---|---|---|
| **0** | 17 | 21903 13107 21463 39275 |
| **1** | 34 | 21137 13176 47766 16083 2596 39475 |
| **2** | 137 | 38689 5134 41787 21903 23794 2326 25890 24852 |
| **3** | 182 | 33000 47672 39275 9337 11520 47209 5479 13629 |
| **4** | 257 | 21137 27966 49235 29837 37646 27104 4605 45013... |
| **...** | ... | ... |
| **74995** | 3420740 | 19660 49005 39928 35951 39146 |
| **74996** | 3420877 | 21137 27966 43122 21903 13176 49111 12238 8277... |
| **74997** | 3420888 | 43961 44632 22935 7963 35951 46906 47766 28985... |
| **74998** | 3420989 | 46676 21616 42450 35004 43210 47766 13517 |
| **74999** | 3421054 | 18426 11123 31231 13375 24852 |

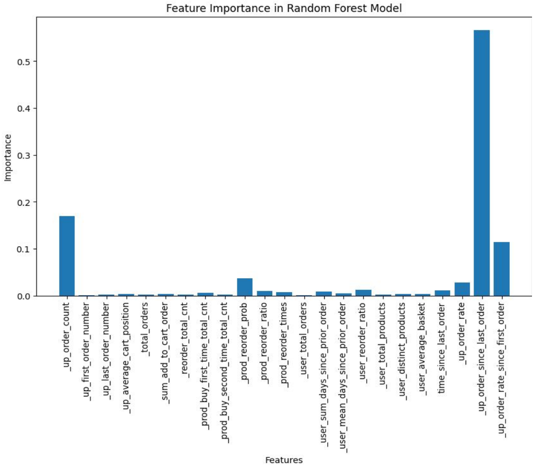
75000 rows × 2 columns

# Feature Importance



# Apply Model

- Perform A/B test before launch
- After launch, continue to monitor model performance (live data support) & business performance
- After launch, continue to collect live traffic data to improve model

# Challenges & Improvement

Challenges

- Big Data Low Ram
- Competion Problem
- Wrong Direction

Improvement

- Thinking More
- Add Better Features
- Try More DL Model
- Try More Ensemble Method
- Keep Study on ML