Project Report

Market Basket Analysis

Group 15
Student 1 Rushiraj Savalia
Student 2 Vishesh Shah

201-919-0757 (Tel of Student 1)
857-800-4630 (Tel of Student 2)

savalia.r@husky.neu.edu
shah.vishe@husky.neu.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Rushiraj Savalia

Signature of Student 2: Vishesh Shah

Submission Date: 04/15/2020

Problem setting

Instacart is an American based company which provides a grocery delivery and pick up service for same day. The company offers its service from website and mobile application in North America. It was founded by a former Amazon employee and entrepreneur Apoorva Mehta in 2012 at San Francisco, California. The application of this project Instacart can regulate their supply chain according to the prediction of this project. Result will give many benefits to supply chain team and will make their customer happy as Instacart can reduce delivery time for their customers. As a team we found many challenges like there is lot of data in this project and how to make sense of it.

Problem Definition

The scope of this project is to predict items of next orders for each user based on the previous orders. There are more than 60,000 customers with their unique user_id.

This project includes following steps.

- Describe various steps of creating a predictive model
- Use R to manipulate data
- Use R to create, combine, and delete data tables
- Use XGBoost algorithm to create a predictive model
- Apply this predictive model in order to make a prediction

Why choose XGBoost?

1. Execution speed
2. Model performance

Data source

Instacart has posted this data for open challenge on Kaggle. We will also post this result on Kaggle after submission.

Data description

There are 6 comma separated files in data.

- orders: This dataset contains all orders, namely prior, train, and test. Primary key (order_id).
- ordert: This dataset contains training orders. Composite primary key (order_id and product_id). Also, it shows that if a product in an order will be reorder or not (reordered variable=0 or 1).
- orderp: This dataset contains prior orders. Composite primary key (order_id and product_id). Also, it shows that if a product in an order will be reorder or not (reordered variable=0 or 1).
- products: This dataset contains all products. Primary key (product_id)
- aisles: This dataset contains all aisles name and aisle_id. Primary key (aisle_id)
- departments: This table includes all departments name and department_id. Primary key (department_id)

The dataset contains various data like aisles name, departments name, customer information, various order of customers. Also, it contains orders from more than 150,000 Instacart user with each user contains unique user_id. Also, each user has their previous order number between 2 to 80 orders. This whole data is divided into three various part prior dataset, train dataset and test dataset. Prior orders show the behavior of a user from past while train and test orders will be used to predict the future behavior of each customers. As a result, we want to predict which products users bought in their prior order and based on that what will be on their next order which can be from train orders or test orders. Each predicted order will be selected based on either train data or test data. This can be described below.

	Order_number									
	1	2	3	4	5	6	7	8	9	10
User A	p	p	p	p	p	tr				
User B	p	p	p	p	p	p	p	p	tr	
User C	p	p	p	p	p	p	te			
User D	p	p	p	p	p	p	p	p	p	tr

Each user has purchased various products during their prior orders. The useful information we have is order_id from previous order which will help us to predict order_id for their future order. The end goad of this project to predict which of these products will be in a user's future order. This can be also viewed as a classification problem because we need to predict whether each pair of user and product is a reorder or not. This is indicated by the value of the reordered variable, i.e. reordered=1 or reordered=0 as shown in figure below.

Instacart users | Products in the prior orders | Future order (train/test) | This is what we need to predict

user_id	product_id	order_id	reordered
1	196	1187899	1
1	10258	1187899	1
1	10326	1187899	0
1	12427	1187899	0
1	13032	1187899	1
1	13176	1187899	0
1	14084	1187899	0
2	17122	2125869	1
2	25133	2125869	0

For prediction we have to calculate various predictors (X) that accurately describes all the characteristics of the data of products and behavior of a user. As a result, we need to come up and calculate various predictors (X) that will describe the characteristics of a product and the behavior of a user regarding one or multiple products. We will do it by analyzing the prior orders from users of the dataset. We will then use the train users to create a predictive model and the test users to make our actual prediction. As a result, we create a table as shown in figure and we train the model using an algorithm that is based on our predictors (X) and response variable (Y).

Primary Key (products from prior orders) | **Predictor variables - X** (based on prior orders) | train/ test | Future order | **Response variable - Y**

user_id	product_id	...	...	...	eval_set	order_id	reordered
1	196				train	1187899	1
1	10258				train	1187899	1
1	10326				train	1187899	0
1	12427				train	1187899	0
1	13032				train	1187899	1
1	13176				train	1187899	0
1	14084				train	1187899	0
2	17122				test	2125869	
2	25133				test	2125869	

Data exploration

We first load the necessary R packages using the install.packages and library() function.

````
```{r}
library(data.table)
library(dplyr)
library(tidyr)
library(Ckmeans.1d.dp)
library(ggplot2)
library(knitr)
library(stringr)
```
````

There are 6 CSV files, which we can load into R using read.csv function.

````
```{r}
setwd("C:/Users/Rushiraj/Desktop/NU/Data mining/Project files")

aisles <- read.csv("aisles.csv", na.strings = "", stringsAsFactors=FALSE)
departments <- read.csv("departments.csv", na.strings = "", stringsAsFactors=FALSE)
orderp <- read.csv("order_products__prior.csv", na.strings = "", stringsAsFactors=FALSE)
ordert <- read.csv("order_products__train.csv", na.strings = "", stringsAsFactors=FALSE)
orders <- read.csv("orders.csv", na.strings = "", stringsAsFactors=FALSE)
products <- read.csv("products.csv", na.strings = "", stringsAsFactors=FALSE)
```
````

Use glimpse() function to check data type of all variables of all tables.

````
```{r}
glimpse(aisles)
glimpse(departments)
glimpse(orderp)
glimpse(orders)
glimpse(ordert)
glimpse(products)
```
````

````
Observations: 134
Variables: 2
$ aisle_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24...
$ aisle    <chr> "prepared soups salads", "specialty cheeses", "energy granola bars", "instant foods",...
Observations: 21
Variables: 2
$ department_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21
$ department    <chr> "frozen", "other", "bakery", "produce", "alcohol", "international", "beverages",...
Observations: 32,434,489
Variables: 4
$ order_id          <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4...
$ product_id        <int> 33120, 28985, 9327, 45918, 30035, 17794, 40141, 1819, 43668, 33754, 24838, 1...
$ add_to_cart_order <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9...
$ reordered         <int> 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1...
Observations: 3,421,083
Variables: 7
$ order_id               <int> 2539329, 2398795, 473747, 2254736, 431534, 3367565, 550135, 3108588, 22...
$ user_id                <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,...
$ eval_set               <chr> "prior", "prior", "prior", "prior", "prior", "prior", "prior", "prior",...
$ order_number           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1...
$ order_dow              <int> 2, 3, 3, 4, 4, 2, 1, 1, 1, 4, 4, 2, 5, 1, 2, 3, 2, 2, 1, 2, 1, 1, 1, 4,...
$ order_hour_of_day      <int> 8, 7, 12, 7, 15, 7, 9, 14, 16, 8, 8, 11, 10, 10, 10, 11, 9, 12, 15, 9, ...
$ days_since_prior_order <dbl> NA, 15, 21, 29, 28, 19, 20, 14, 0, 30, 14, NA, 10, 3, 8, 8, 13, 14, 27,...
Observations: 1,384,617
Variables: 4
$ order_id          <int> 1, 1, 1, 1, 1, 1, 1, 1, 36, 36, 36, 36, 36, 36, 36, 36, 38, 38, 38, 38, 38, ...
$ product_id        <int> 49302, 11109, 10246, 49683, 43633, 13176, 47209, 22035, 39612, 19660, 49235,...
$ add_to_cart_order <int> 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1...
$ reordered         <int> 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1...
Observations: 49,688
Variables: 4
$ product_id    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2...
$ product_name  <chr> "Chocolate Sandwich Cookies", "All-Seasons Salt", "Robust Golden Unsweetened Ool...
$ aisle_id      <int> 61, 104, 94, 38, 5, 11, 98, 116, 120, 115, 31, 119, 11, 74, 56, 103, 35, 79, 63,...
$ department_id <int> 19, 13, 7, 1, 13, 11, 7, 1, 16, 7, 7, 1, 11, 17, 18, 19, 12, 1, 9, 7, 8, 11, 12,...
````

We now use the head() function in order to visualize the first 10 rows of these tables.

```{r}
head(aisles,10)
head(departments,10)
head(orderp,10)
head(orders,10)
head(ordert,10)
head(products,10)
```

data.frame 10 x 2 | data.frame 10 x 2 | data.frame 10 x 4 | data.frame 10 x 7 | data.frame 10 x 4 | data.frame 10 x 4

	product_id <int>	product_name <chr>	aisle_id <int>
1	1	Chocolate Sandwich Cookies	61
2	2	All-Seasons Salt	104
3	3	Robust Golden Unsweetened Oolong Tea	94
4	4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce	38
5	5	Green Chile Anytime Sauce	5
6	6	Dry Nose Oil	11
7	7	Pure Coconut Water With Orange	98
8	8	Cut Russet Potatoes Steam N' Mash	116
9	9	Light Strawberry Blueberry Yogurt	120
10	10	Sparkling Orange Juice & Prickly Pear Beverage	115

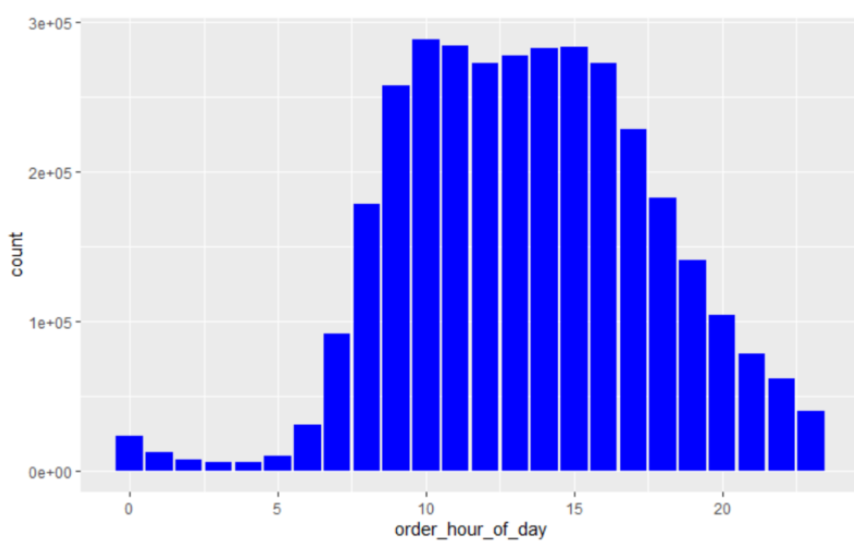
1-10 of 10 rows | 1-4 of 4 columns

Data Visualization

When do People order groceries

```{r}
#When people buy groceries online.

orders %>%
  ggplot(aes(x=order_hour_of_day)) +
  geom_histogram(stat="count",fill="blue")

#Hour of Day
#There is a clear effect of hour of day on order volume. Most orders are between 8.00-18.00
```

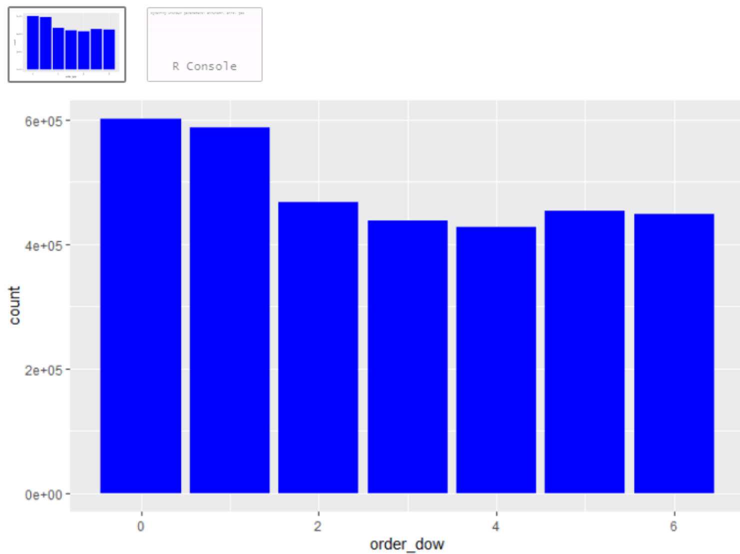
R Console



From graph we can see that people mostly order between 8 am to 6 pm.

On which days people order

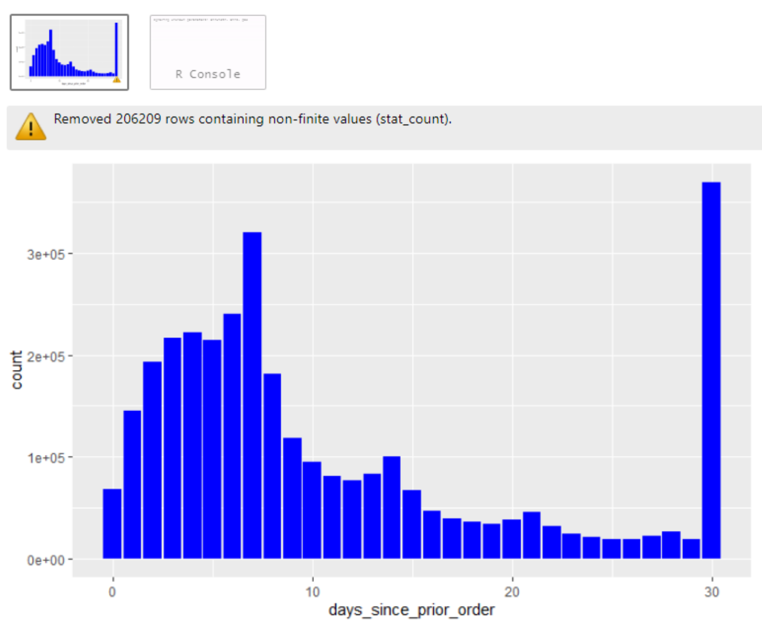
```{r}
orders %>%
  ggplot(aes(x=order_dow)) +
  geom_histogram(stat="count",fill="blue")
#Most orders are on days 0 and 1 which is Sunday and Monday.
```



From the graph we can see that people order on Sunday and Monday.

When do people order again

```{r}
#When do they order again?

orders %>%
  ggplot(aes(x=days_since_prior_order)) +
  geom_histogram(stat="count",fill="blue")

#People seem to order more often after exactly 1 week.
```



From graph we can see that people order again in 1 week.

Data mining tasks

This method includes the following steps:

1. Import and reshape data: This step includes loading CSV files into R tables, transform character variables to categorical variables, and create a supportive table.
2. Calculate predictor variables: This step includes identifying and calculating predictor variables from the data.
3. Create the test and train datasets: This step includes partition into test and train dataset from the data we have
4. Create the predictive model: This step includes applying XGBoost algorithm to create the predictive model through the train dataset.
5. Apply the model: This step includes applying the model to predict the 'reordered' variable for the test dataset.

Reshape data

We transform the data according to our need and which can be used for ease operation for analysis. Convert all character variables into factors because factors can be used for analysis. Also, we will use head() function after each iteration as we can see the table and first 10 records.

````
```{r}
#Convert all categorial variable to factor as we can use them to make better model
aisles$aisle <- as.factor(aisles$aisle)
departments$department <- as.factor(departments$department)
orders$eval_set <- as.factor(orders$eval_set)
products$product_name <- as.factor(products$product_name)
```
````

We replace aisle_id with aisle name and department_id with department name in products table.

```{r}
#In the products table, replace aisle_id and department_id with aisle name and department name
products <- products %>%
  inner_join(aisles) %>%
  inner_join(departments) %>%
  select(-aisle_id, -department_id)

#Now we don't need ailes and departments so we will remove it.
rm(aisles, departments)

#New products table
head(products,10)
```

R Console

data.frame
10 x 4

	product_id <int>	product_name <fctr>
1	1	Chocolate Sandwich Cookies
2	2	All-Seasons Salt
3	3	Robust Golden Unsweetened Oolong Tea
4	4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce
5	5	Green Chile Anytime Sauce
6	6	Dry Nose Oil
7	7	Pure Coconut Water With Orange
8	8	Cut Russet Potatoes Steam N' Mash
9	9	Light Strawberry Blueberry Yogurt
10	10	Sparkling Orange Juice & Prickly Pear Beverage

1-10 of 10 rows | 1-3 of 4 columns

Then we add user_id column from ordert where order_id match in ordert and orders table.

```{r}
#Add the column user_id to model
ordert$user_id <- orders$user_id[match(ordert$order_id, orders$order_id)]

#Check model
head(ordert,10)
```

	order_id <int>	product_id <int>	add_to_cart_order <int>	reordered <int>	user_id <int>
1	1	49302	1	1	112108
2	1	11109	2	1	112108
3	1	10246	3	0	112108
4	1	49683	4	0	112108
5	1	43633	5	1	112108
6	1	13176	6	0	112108
7	1	47209	7	0	112108
8	1	22035	8	1	112108
9	36	39612	1	0	79431
10	36	19660	2	1	79431

1-10 of 10 rows

Create an orders_products table

We create a new table orders_products which contains the tables orders and ordersp, which can be done using inner_join() function. This function only returns record that have matching values in both tables.

````
```{r}
#Create a new table orders_products which contains the tables "orders" and orderp to make model
orders_products <- orders %>% inner_join(orderp, by = "order_id")

#remove orderp
rm(orderp)
#clear memory for smooth process
gc()

#Make new orders_products table
head(orders_products,10)
```
````

R Console

data.frame
10 x 10

	order_id <int>	user_id <int>	eval_set <fctr>	order_number <int>	order_dow <int>	order_hour_of_day <int>
1	2539329	1	prior	1	2	8
2	2539329	1	prior	1	2	8
3	2539329	1	prior	1	2	8
4	2539329	1	prior	1	2	8
5	2539329	1	prior	1	2	8
6	2398795	1	prior	2	3	7
7	2398795	1	prior	2	3	7
8	2398795	1	prior	2	3	7
9	2398795	1	prior	2	3	7
10	2398795	1	prior	2	3	7

1-10 of 10 rows | 1-7 of 10 columns

Create Predictor Variables

Predictors from Product table (prd table)

Create prd table contains predictors that represents Product table.

1. prod_orders: Total number of orders per product
   prod_orders= prod_first_orders*prod_reorder_times
   prod_reorder_times= 1+ (prod_reorders/prod_first_orders)
2. prod_reorder_probability: Probability a product is reordered after the first order
   prod_reorder_probability= prod_second_orders/prod_first_orders
3. prod_reorder_times: How many times a product has been purchased by the users
4. prod_reorder_ratio: Reorders per total number of orders of the product

To calculate these predictor variables, we must calculate supporting variables

prod_reorders: Total number of reorders per product

prod_first_orders: Total number of first orders per product

prod_second_orders: Total number of second orders per product

Created temporary prd table

```r
# We create the prd and we start with the data inside the orders_products table
#Create temporary prd to create model which contains data from order_products
prd <- orders_products %>%
arrange(user_id, order_number, product_id) %>%
group_by(user_id, product_id) %>%
mutate(product_time = row_number()) %>%          #Create the new variable product time through row_number()
ungroup()                                         #Identified how many times a user bought a product

#See the temporary prd table
head(prd,10)
```

order_id <int>	user_id <int>	eval_set <fctr>	order_number <int>	order_dow <int>	order_hour_of_day <int>	days_since_prior_order <dbl>
2539329	1	prior	1	2	8	NA
2539329	1	prior	1	2	8	NA
2539329	1	prior	1	2	8	NA
2539329	1	prior	1	2	8	NA
2539329	1	prior	1	2	8	NA
2398795	1	prior	2	3	7	15
2398795	1	prior	2	3	7	15
2398795	1	prior	2	3	7	15
2398795	1	prior	2	3	7	15
2398795	1	prior	2	3	7	15

1-10 of 10 rows | 1-7 of 11 columns

Update temporary prd table by calculating supporting variables

```r
prd <- prd %>%
group_by(product_id) %>%                      #Group by product_id
summarise(
    prod_orders = n(),                        #Total number of orders per product
    prod_reorders = sum(reordered),           #Total number of reorders per product
    prod_first_orders = sum(product_time == 1),
    prod_second_orders = sum(product_time == 2))

#See the temporary prd table
head(prd,10)
```

product_id <int>	prod_orders <int>	prod_reorders <int>	prod_first_orders <int>	prod_second_orders <int>
1	1852	1136	716	276
2	90	12	78	8
3	277	203	74	36
4	329	147	182	64
5	15	9	6	4
6	8	3	5	2
7	30	12	18	6
8	165	83	82	30
9	156	82	74	31
10	2572	1304	1268	399

1-10 of 10 rows

Calculate final product predictors and make final prd table

```{r}
#Calculate prod_reorder_probability variable
prd$prod_reorder_probability <- prd$prod_second_orders / prd$prod_first_orders
#Caclculate the prod_reorder_times variable
prd$prod_reorder_times <- 1 + prd$prod_reorders / prd$prod_first_orders
#Caclculate the prod_reorder_ratio variable
prd$prod_reorder_ratio <- prd$prod_reorders / prd$prod_orders

#Remove the prod_reorders, prod_first_orders, and prod_second_orders variables
prd <- prd %>% select(-prod_reorders, -prod_first_orders, -prod_second_orders)

#Remove products table
rm(products)
#clear memory for smooth process
gc()

#See the final prd table
head(prd,20)
```

R Console

tbl_df
20 x 5

product_id <int>	prod_orders <int>	prod_reorder_probability <dbl>	prod_reorder_times <dbl>	prod_reorder_ratio <dbl>
1	1852	0.38547486	2.586592	0.6133909
2	90	0.10256410	1.153846	0.1333333
3	277	0.48648649	3.743243	0.7328520
4	329	0.35164835	1.807692	0.4468085
5	15	0.66666667	2.500000	0.6000000
6	8	0.40000000	1.600000	0.3750000
7	30	0.33333333	1.666667	0.4000000
8	165	0.36585366	2.012195	0.5030303
9	156	0.41891892	2.108108	0.5256410
10	2572	0.31466877	2.028391	0.5069984

1-10 of 20 rows Previous 1 2 Next

Predictors from Users table (users table)

Update users table by calculating new predictor variables from users table. For this we only use data from prior orders.

1. user_orders: Total number of orders per user
2. user_period: The time between the first and last order of a user
3. user_mean_days_since_prior: Mean time between two consecutive orders of a single user

```{r}
users <- orders %>%
  filter(eval_set == "prior") %>%                  #Keep only prior order
  group_by(user_id) %>%                            #Group orders by user_id
  summarise(                                       #Total number of orders per user
    user_orders = max(order_number),               #Omit the missing values and calculate the sum and mean
    user_period = sum(days_since_prior_order, na.rm = T),
    user_mean_days_since_prior = mean(days_since_prior_order, na.rm = T))

#See the temporary users table
head(users,10)
```

user_id <int>	user_orders <int>	user_period <dbl>	user_mean_days_since_prior <dbl>
1	10	176	19.55556
2	14	198	15.23077
3	12	133	12.09091
4	5	55	13.75000
5	4	40	13.33333
6	3	18	9.00000
7	20	203	10.68421
8	3	60	30.00000
9	3	36	18.00000
10	5	79	19.75000

1-10 of 10 rows

Created new us table that contains new variables

- user_total_products: Total numbers of basket items included in user's orders
- user_reorder_ratio: Reorder ratio per user
- user_distinct_products: Total number of distinct products ordered by a user

Group these observations by user_id from orders_products table.

```
```{r}
us <- orders_products %>%
  group_by(user_id) %>%
  summarise(
    user_total_products = n(),
    user_reorder_ratio = sum(reordered == 1) / sum(order_number > 1),
    user_distinct_products = n_distinct(product_id) #Counts the number of unique values in model
    )

#See the us table
head(us,10)
```
```

user_id <int>	user_total_products <int>	user_reorder_ratio <dbl>	user_distinct_products <int>
1	59	0.75925926	18
2	195	0.51098901	102
3	88	0.70512821	33
4	18	0.07142857	17
5	37	0.53846154	23
6	14	0.20000000	12
7	206	0.71134021	68
8	49	0.46428571	36
9	76	0.39130435	58
10	143	0.35507246	94

1-10 of 10 rows

Combine users and us tables using inner_join() function and calculate the final variable:

user_average_basket: Average number of basket items per order per user

```
```{r}
#Combine users and us tables and store the results into users table
users <- users %>% inner_join(us)

#Calculate the user_average_basket variable
users$user_average_basket <- users$user_total_products / users$user_orders

#See the users table
head(users,10)
```
```

R Console

tbl_df
10 x 8

user_id <int>	user_orders <int>	user_period <dbl>	user_mean_days_since_prior <dbl>	user_total_products <int>
1	10	176	19.55556	59
2	14	198	15.23077	195
3	12	133	12.09091	88
4	5	55	13.75000	18
5	4	40	13.33333	37
6	3	18	9.00000	14
7	20	203	10.68421	206
8	3	60	30.00000	49
9	3	36	18.00000	76
10	5	79	19.75000	143

1-10 of 10 rows | 1-5 of 8 columns

Now, we will find future order for each user and update it in users table. The future orders are indicated as train and test in the eval_set variable. The main reason of doing so is that we will know what is the order_id of the future order per user, whether this order belongs in the train or test set, and the time in days since the last order.

```{r}
us <- orders %>%
  filter(eval_set != "prior") %>%          #Exclude prior orders and keep only train and test order
  select(user_id, order_id, eval_set,time_since_last_order = days_since_prior_order)

#Combine users and us tables and store the results into the users table
users <- users %>% inner_join(us)

#Remove the us table
rm(us)
#Garbage collection. clear memory for smooth process
gc()

#See the final users table
head(users,10)
```

R Console

tbl_df
10 x 11

user_id <int>	user_orders <int>	user_period <dbl>	user_mean_days_since_prior <dbl>	user_total_products <int>
1	10	176	19.55556	59
2	14	198	15.23077	195
3	12	133	12.09091	88
4	5	55	13.75000	18
5	4	40	13.33333	37
6	3	18	9.00000	14
7	20	203	10.68421	206
8	3	60	30.00000	49
9	3	36	18.00000	76
10	5	79	19.75000	143

1-10 of 10 rows | 1-5 of 11 columns

New predictors from Users table (data table)

We created data table and store predictors that shows how a user behaves for a single product. We use prd and users table to create data table.

1. up_orders: The total times a user ordered a product
2. up_first_order: What was the first time a user purchased a product
3. up_last_order: What was the last time a user purchased a product
4. up_average_cart_position: The average position in a user's cart of a product
5. up_order_rate: Percentage of user’s orders that include a specific product
6. up_orders_since_last_order: Number of orders since user’s last order of a product
7. up_order_rate_since_first_order: Pecentage of orders since first order of a product in which a user purchased this product

```{r}
data <- orders_products %>%
  group_by(user_id, product_id) %>%
  summarise(
    up_orders = n(),
    up_first_order = min(order_number),
    up_last_order = max(order_number),
    up_average_cart_position = mean(add_to_cart_order))

#Remove the tables orders_products and orders
rm(orders_products, orders)

#See the temporary data table
head(data, 10)
```

user_id <int>	product_id <int>	up_orders <int>	up_first_order <int>	up_last_order <int>	up_average_cart_position <dbl>
1	196	10	1	10	1.400000
1	10258	9	2	10	3.333333
1	10326	1	5	5	5.000000
1	12427	10	1	10	3.300000
1	13032	3	2	10	6.333333
1	13176	2	2	5	6.000000
1	14084	1	1	1	2.000000
1	17122	1	5	5	6.000000
1	25133	8	3	10	4.000000
1	26088	2	1	2	4.500000

1-10 of 10 rows

Combine data table with prd and users to calculate remaining predictors

```{r}
#use inner_join to combine the table data with the tables prd and users
data <- data %>%
  inner_join(prd, by = "product_id") %>%
  inner_join(users, by = "user_id")

#Calculate up_order_rate, up_orders_since_last_order, up_order_rate_since_first_order
data$up_order_rate <- data$up_orders / data$user_orders
data$up_orders_since_last_order <- data$user_orders - data$up_last_order
data$up_order_rate_since_first_order <- data$up_orders / (data$user_orders - data$up_first_order + 1)

#See the temporary data table
head(data, 10)
```

user_id <int>	product_id <int>	up_orders <int>	up_first_order <int>	up_last_order <int>	up_average_cart_position <dbl>
1	196	10	1	10	1.400000
1	10258	9	2	10	3.333333
1	10326	1	5	5	5.000000
1	12427	10	1	10	3.300000
1	13032	3	2	10	6.333333
1	13176	2	2	5	6.000000
1	14084	1	1	1	2.000000
1	17122	1	5	5	6.000000
1	25133	8	3	10	4.000000
1	26088	2	1	2	4.500000

1-10 of 10 rows | 1-6 of 23 columns

Combine the data table with ordert table to find which products user has already bought and recorded.

This can be done using leftjoin() function. If a product matches to ordert table from data table reorderd=1 will saved else reorderd=0.

```{r}
data <- data %>%
  left_join(ordert %>% select(user_id, product_id, reordered),
            by = c("user_id", "product_id"))

#Remove the tables ordert, prd, users
rm(ordert, prd, users)
#Garbage collection. clear memory for smooth process
gc()

#See the final data table. Final model
head(data, 10)
```

R Console

grouped_df
10 x 24

user_id <int>	product_id <int>	up_orders <int>	up_first_order <int>	up_last_order <int>	up_average_cart_position <dbl>
1	196	10	1	10	1.400000
1	10258	9	2	10	3.333333
1	10326	1	5	5	5.000000
1	12427	10	1	10	3.300000
1	13032	3	2	10	6.333333
1	13176	2	2	5	6.000000
1	14084	1	1	1	2.000000
1	17122	1	5	5	6.000000
1	25133	8	3	10	4.000000
1	26088	2	1	2	4.500000

1-10 of 10 rows | 1-6 of 24 columns

Create the Train & Test tables

Create final train and test tables using all the data we have and then apply the XGBoost algorithm.

This process includes the following steps.

- Split the data based on the eval_set variable into train and test.
- Remove all the columns that are not predictors variables.
- In the train set we transform NA values of reordered to 0 in order to indicate that these products have not been reordered in the future order.

Training Model

```{r}
#Training Model
train <- as.data.frame(data[data$eval_set == "train",])
train$eval_set <- NULL
train$user_id <- NULL
#Transform missing values of reordered variable to 0
train$reordered[is.na(train$reordered)] <- 0

#See train table
head(train,10)
```

	product_id <int>	up_orders <int>	up_first_order <int>	up_last_order <int>	up_average_cart_position <dbl>	prod_orders <int>
1	196	10	1	10	1.400000	35791
2	10258	9	2	10	3.333333	1946
3	10326	1	5	5	5.000000	5526
4	12427	10	1	10	3.300000	6476
5	13032	3	2	10	6.333333	3751
6	13176	2	2	5	6.000000	379450
7	14084	1	1	1	2.000000	15935
8	17122	1	5	5	6.000000	13880
9	25133	8	3	10	4.000000	6196
10	26088	2	1	2	4.500000	2523

1-10 of 10 rows | 1-7 of 22 columns

Testing Model

```{r}
#Testing Model
test <- as.data.frame(data[data$eval_set == "test",])
test$eval_set <- NULL
test$user_id <- NULL
test$reordered <- NULL

#See test table
head(test,10)
```

	product_id <int>	up_orders <int>	up_first_order <int>	up_last_order <int>	up_average_cart_position <dbl>	prod_orders <int>
1	248	1	2	2	3.000000	6371
2	1005	1	10	10	5.000000	463
3	1819	3	4	7	2.666667	2424
4	7503	1	3	3	6.000000	12474
5	8021	1	2	2	5.000000	27864
6	9387	5	1	7	3.600000	36187
7	12845	1	4	4	2.000000	10027
8	14992	2	6	7	7.000000	29069
9	15143	1	1	1	3.000000	3447
10	16797	3	1	9	4.000000	142951

1-10 of 10 rows | 1-7 of 21 columns

Data mining models/methods

We use the train data to create the model. Each variable is a list containing two things, outcome and data. We want to predict is the column reordered. Because xgboost manages only numeric vectors we have to transform the categorical data to dummy variables while building the model we will need.

parameters of algorithm

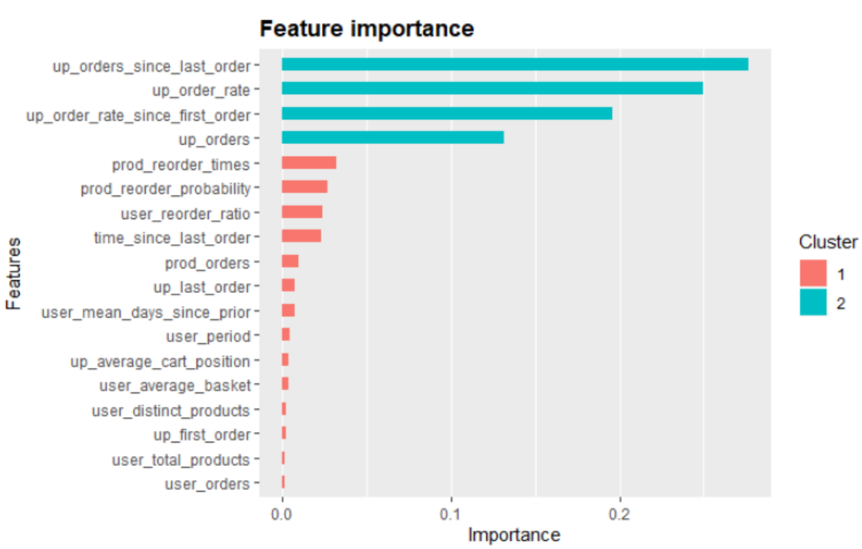
1. objective: logistic regression
2. eval_metric: logloss function
3. eta: default 0.1. Range 0 to 1.
4. max_depth: The default value is set to 6. You need to specify the maximum depth or splits of a tree. The range is 1 to ∞
5. min_child_weight: Default 1. Range to infinity.
6. gamma: Default 0. Range 0 to infinity.
7. subsample: Default 1. Range 0 to 1
8. colsample_bytree: Default 1. Range 0 to 1
9. alpha: These are regularization term on weights. Alpha default value assumed is 0
10. lambda: Weights. Default value 1
11. nround: The number of trees to the model

Another benefit that gradient boosting provide is they automatically provides estimates of feature importance from trained predictive model. Feature importance in Xgboost can be done through xgb.importance(). Final result of this is a table that contains all variables, their gain and frequency. Gain is the improvement in accuracy brought by a feature to the branches it is on. Cover measures the relative quantity of observations concerned by a feature. Frequency is a simpler way to measure the Gain. The column Gain provide the information we are looking for. Finally we can plot the results from our model using xgb.plot.importance() function. The feature importance graph is showing below.

```{r}
library(xgboost)

params <- list(
  "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
)

#Sampling technique. 10% of the train table
subtrain <- train %>% sample_frac(0.1)
#Create an xgb.DMatrix that is named X with predictors from subtrain table and response the reordered variable
X <- xgb.DMatrix(as.matrix(subtrain %>% select(-reordered, -order_id, -product_id)), label = subtrain$reordered
#Create the actual model
model <- xgboost(data = X, params = params, nrounds = 80)
```

```
package �xgboost� was built under R version 3.6.3
Attaching package: �xgboost�

The following object is masked from �package:dplyr�:

    slice

[1]	train-logloss:0.626004
[2]	train-logloss:0.570561
[3]	train-logloss:0.525036
[4]	train-logloss:0.486456
[5]	train-logloss:0.453949
[6]	train-logloss:0.426235
[7]	train-logloss:0.402462
[8]	train-logloss:0.382085
[9]	train-logloss:0.364503
[10]	train-logloss:0.349226
[11]	train-logloss:0.336008
[12]	train-logloss:0.324499
[13]	train-logloss:0.314394
[14]	train-logloss:0.305754
```

```{r}
#Estimate the importance of the predictors
importance <- xgb.importance(colnames(X), model = model)
#Plot the importance of the predictors
xgb.ggplot.importance(importance)

rm(X, importance, subtrain)
gc()
```



Performance evaluation

Apply the model to test data

Now we are ready to perform the prediction with the model we have built to classify test data.

```{r}
#Use the xgb.DMatrix to group our test data into a matrix
X <- xgb.DMatrix(as.matrix(test %>% select(-order_id, -product_id)))
#Apply the model and we predict the reordered variable for the test set.
test$reordered <- predict(model, X)
#The model estimates a probability.
#Apply a threshold so every prediction above 0.21 will be considered as a reorder (reordered=1)
test$reordered <- (test$reordered > 0.21) * 1

#Create the final table with reordered products per order
submission <- test %>%
              filter(reordered == 1) %>%
              group_by(order_id) %>%
              summarise(products = paste(product_id, collapse = " "))
#submission table
head(submission,10)
```

order_id <int>
17
34
137
182
257
313
353
386
414
418

1-10 of 10 rows | 1-1 of 2 columns

```{r}
#Create the table missing where we have the orders in which none product will be ordered according to our
prediction
missing <- data.frame(
                      order_id = unique(test$order_id[!test$order_id %in% submission$order_id]),
                      products = "None"
                     )

submission <- submission %>% bind_rows(missing) %>% arrange(order_id)
#See the submission table
head(submission,10)
```

R Console

tbl_df
10 x 2

order_id <int>
17
34
137
182
257
313
353
386
414
418

1-10 of 10 rows | 1-1 of 2 columns

In order to be able to compare our prediction to the actual result we re-apply the model in the train data.

Apply the model in train data

Now we are ready to perform the prediction with the model we have built to classify train data.

```
```{r}
#Use the xgb.DMatrix to group our train data into a matrix
X <- xgb.DMatrix(as.matrix(train %>% select(-order_id, -product_id, -reordered)))
#Apply the model and we predict the reordered variable for the train set.
train$reordered_pred <- predict(model, X)
#The model estimates a probability.
#Apply a threshold so every prediction above 0.21 will be considered as a reorder (reordered=1)
train$reordered_pred <- (train$reordered_pred > 0.21) * 1

#Create the final table with reordered products per order
submission_train <- train %>%
  filter(reordered_pred == 1) %>%
  group_by(order_id) %>%
  summarise(products = paste(product_id, collapse = " "))

real_reorders <- train %>%
 filter(reordered == 1) %>%
 group_by(order_id) %>%
 summarise(
   real_products = paste(product_id, collapse = " "))

submission_train <- real_reorders %>%
  inner_join(submission_train, by = "order_id")

#See the submission table
head(submission_train,10)
```
```

order_id <int>
1
36
38
96
98
112
170
218
349
393

1-10 of 10 rows | 1-1 of 3 columns

real_products <chr>
11109 22035 43633 49302
19660 34497 43086 46620 46979 48679
21616
20574 24489 27966 39275 40706
329 790 1939 3880 4357 7461 8859 9373 9896 13176 15455 15995 17747 18117 18441 19731 20520 22935 24964 26...
5876 21174 27104
5077 6236 13176 18394 37766 40354
1194 5578
5115 11361 11520 27695 33000
6184 12078 13424 16797 19828 30591 32403

1-10 of 10 rows | 2-2 of 3 columns

products <chr>
5707 11109 14947 22035 24852 30881 43633 44359 44632 49302
19660 24964 38293 44359
8012 33731
20574 24489 27966 29603
329 790 1939 3339 3880 4357 5451 8518 8859 9373 9896 13176 15455 15995 18117 19731 21616 22963 24964 27344 2...
3599 4799 5646 5785 5876 13176 18070 21174 22935 24964 25199 25472 27104 34243 35121 40821 46880 47119 47766
5077 5223 6236 13176 18394 19953 25748 25804 28092 34789 35124 37766
1194 5578 15763 19505
5115 6258 10369 11361 11520 16349 19862 21982 27695 32864 33000 33198 34214 37665
1689 6184 8048 12078 16797 19828 21288 30591 32403

1-10 of 10 rows | 3-3 of 3 columns

Project results

We can see that when we apply the model to test and train data it shows order_id having that number can order the products as shown in second column as shown in figure.

Insights for decision making

Based on the list we got in both tables, Instacart can already order those products in advance and save supply chain cost.

Impact of the project outcomes

We can submit these results to the officials of Instacart. Also, another benefit is that they can deliver all things while it's fresh. That will surely make their customers happy and grow business.