CYO edX Capstone Project – Instacart/Market Basket Kaggle Data Set

Marina Yamasaki

1/8/2021

1. Intro/Overview/Exec Summary: (describes dataset, summarizes goals of project and key steps performed)

Instacart/Market Basket 2017 Dataset: https://www.kaggle.com/c/instacart-market-basket-analysis Instacart is a service that enables consumers to purchase groceries without going to the grocery store. A consumer will use the instacart portal/app to purchase items and an instacart delivery employee will pick up and deliver those items to the consumer's home. In 2017, Instacart released a dataset on kaggle challenging the public to build models that could predict the products that an Instacart consumer might purchase in their next order. I have chosen to evaluate the Instacart data for my CYO Capstone project to continue to experiment with different machine learning techniques and further my overall data science understanding. The data provided by Instacart consists of 6 files. The products, departments, and aisles files are reference tables, they provide names/descriptions for the product, department, and aisle IDs. The orders data file contains ~3.4M grocery orders and ~200K users. Each order is tied to a specific user. The order dataset also identifies how many orders have been made by each user and on what day of the week. The order_products_prior and order_products_train data tables breakout the products purchased in each order. The order_products_prior/train tables provide more insight into a users order history and product preferences. For instance the tables indicate if a product purchase is a reorder, what order the products were added to the cart, and how many days passed between a users orders. The order_product_train table only contains information about a user's most recent order. After downloading and cleaning the provided data tables, exploratory data analysis will be conducted to understand the data. Visualizations will be constructed to understand product popularity, user purchase behavior, and other notable trends. As the data is separated between various tables, work was done to parse out relevant data fields, aggregate information to build predictive variables, and breakout the data for variable construction, model development, and model testing.

Data Preparation: Install relevant packages and libraries:

```
if(!require(dplyr))
  install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(lubridate))
  install.packages('lubridate', repos = "http://cran.us.r-project.org")
if(!require(recosystem))
  install.packages('recosystem', repos = "http://cran.us.r-project.org")
```

```
if(!require(rmarkdown))
  install.packages('rmarkdown', repos = "http://cran.us.r-project.org")
if(!require(rpart))
  install.packages('rpart', repos = "http://cran.us.r-project.org")
if(!require(randomForest))
  install.packages('randomForest', repos = "http://cran.us.r-project.org")
if(!require(xgboost))
  install.packages("xgboost", repos = "http://cran.us.r-project.org")
if(!require(Ckmeans.1d.dp))
  install.packages("xCkmeans.1d.dp", repos = "http://cran.us.r-project.org")
if(!require(googledrive))
  install.packages("googledrive", repos = "http://cran.us.r-project.org")
if(!require(kableExtra))
  install.packages('kableExtra', repos = "http://cran.us.r-project.org")
library('dplyr')
library('tidyverse')
library('caret')
library('data.table')
library('stringr')
library('lubridate')
library('rmarkdown')
library('recosystem')
library('rmarkdown')
library('rpart')
library('randomForest')
library('xgboost')
library('Ckmeans.1d.dp')
library('googledrive')
library('kableExtra')
```

The instacart/market basket dataset is loaded into the following google drive folder: https://drive.google.com/drive/folders/1p-A5scQDoUHygHhpPBJvyZg03WnF1_6d?usp=sharing. The departments, aisles, products, and order_products___train data tables will be automatically downloaded from the google drive. The orders and order_products___prior will need to be manually downloaded (because of their file size, google drive requires user permission to download the data). The following code will unzip and read in the orders and order_products___prior data tables.

```
dep_id <- "1cutfNKqPkJ0bxRuCaIVZJ9G5Bi7h95l0"
ais_id <- "1sW6yTGC_y71mrFmZqkLFO8LGC9pupHuw"
prod_id <- "1nPpK8ICihE--rjQ0Yvr-_C2QVeaEeowT"
opt_id <- "124_XGBr2qZ67Ef24DR7I_yFnohmR4emt"

departments <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", dep_id))
aisles <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", ais_id))
products <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", prod_id))
order_products_train <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", opt_id))

getwd()
```

```
## [1] "/Users/my_saki/Documents/R Coding/edX Capstone Project/Instacart Market Basket Data Set"
```

```
# setwd("./Instacart Market Basket Data Set")
orders <- read.csv(unzip("./orders.zip","orders.csv"))
order_products_prior <- read.csv(unzip("./order_products__prior.zip","order_products__prior.csv"))
```

Set the categorical data as factors:

```
aisles <- aisles %>% mutate(aisle = as.factor(aisle))
departments <- departments %>% mutate(department = as.factor(department))
products <- products %>% mutate(product_name = as.factor(product_name))
orders <- orders %>% mutate(eval_set = as.factor(eval_set))
```

Data Overview: The data contains 49,688 products

```
head(products)
```

```
##   product_id                                                  product_name
## 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
##   aisle_id department_id
## 1       61            19
## 2      104            13
## 3       94             7
## 4       38             1
## 5        5            13
## 6       11            11
```

Products are organized within 134 aisles

```
head(aisles)
```

```
##   aisle_id                       aisle
## 1        1      prepared soups salads
## 2        2          specialty cheeses
## 3        3        energy granola bars
## 4        4              instant foods
## 5        5 marinades meat preparation
## 6        6                      other
```

Products are classified within 21 departments

```
head(departments)
```

```
##   department_id    department
## 1             1        frozen
## 2             2         other
## 3             3        bakery
## 4             4       produce
## 5             5       alcohol
## 6             6 international
```

There are 7 variables in the orders data set: order_id and user_id are table keys. Each order has a distinct order_id and each user has a distinct user_id. eval_set consists of 3 values - prior, train, and test. The latest order for any user is classified as either train or test. Any previous order for a user is classified as prior. As a result, the prior orders will be used to develop predictive variables to determine if a product will be reordered. The train orders will be partitioned. A large part of the train orders will be used to develop the models and a portion of the train orders will be used to validate the models. order_dow - indicates on what day of the week the order was placed order_hour_of_day - indicates which hour of the day an order was placed days_since_prior - indicates the number of days between a user's consecutive orders, value is null for the user's first order

```
glimpse(orders)
```

```
## Rows: 3,421,083
## Columns: 7
## $ order_id              <int> 2539329, 2398795, 473747, 2254736, 431534, 3...
## $ user_id               <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,...
## $ eval_set              <fct> prior, prior, prior, prior, prior, prior, pr...
## $ order_number          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1, 2, 3, ...
## $ order_dow             <int> 2, 3, 3, 4, 4, 2, 1, 1, 1, 4, 4, 2, 5, 1, 2,...
## $ order_hour_of_day     <int> 8, 7, 12, 7, 15, 7, 9, 14, 16, 8, 8, 11, 10,...
## $ days_since_prior_order <dbl> NA, 15, 21, 29, 28, 19, 20, 14, 0, 30, 14, N...
```

Examining the orders specific to user 1 illustrates the data structure. User 1's latest order is categorized as eval_set train. All prior orders are categorized as 'prior'. The data is partitioned so that the latest order for every user_id is categorized as either 'test' or 'train'. User 100's latest order has been categorized as test. In total 75K orders (~36% of latest orders) have been categorized as test. 131K orders are in the train set.

```
orders %>% filter(user_id == 1) # %>% kable %>% kable_styling(latex_options="scale_down")
```

```
##    order_id user_id eval_set order_number order_dow order_hour_of_day
## 1   2539329       1    prior            1         2                 8
## 2   2398795       1    prior            2         3                 7
## 3    473747       1    prior            3         3                12
## 4   2254736       1    prior            4         4                 7
## 5    431534       1    prior            5         4                15
## 6   3367565       1    prior            6         2                 7
## 7    550135       1    prior            7         1                 9
## 8   3108588       1    prior            8         1                14
## 9   2295261       1    prior            9         1                16
## 10  2550362       1    prior           10         4                 8
## 11  1187899       1    train           11         4                 8
##    days_since_prior_order
## 1                      NA
## 2                      15
## 3                      21
## 4                      29
## 5                      28
## 6                      19
## 7                      20
## 8                      14
## 9                       0
## 10                     30
## 11                     14
```

```
orders %>% filter(user_id == 100) # %>% kable %>% kable_styling(latex_options="scale_down")
```

```
##   order_id user_id eval_set order_number order_dow order_hour_of_day
## 1   680467     100    prior            1         3                18
## 2  3159209     100    prior            2         0                15
## 3  2443738     100    prior            3         5                17
## 4  2337051     100    prior            4         3                20
## 5  2875733     100    prior            5         1                18
## 6  3302990     100     test            6         5                19
##   days_since_prior_order
## 1                     NA
## 2                     18
## 3                     30
## 4                     30
## 5                     26
## 6                     30
```

```
length(orders$eval_set[orders$eval_set == 'train'])
```

```
## [1] 131209
```

```
length(orders$eval_set[orders$eval_set == 'test'])
```

```
## [1] 75000
```

The order_products_prior/train data tables contain 4 columns: order_id, product_id, add_to_cart_order, and reordered. These tables provide more detail to the orders table by breaking out the products purchased within each order. The add_to_cart_order variable indicates the order in which a product was added to the online cart. The reordered variable is a binary indicator, 1 means the order has been purchased by the user before, 0 means the product is being purchased for the first time by the user via Instacart. Reordered is our response variable for this analysis. Based on characteristics and variables of the user, products, user's past buying behavior... models will be built to predict if an item is reordered. The order_products_prior table contains 32.4M rows

```
glimpse(order_products_prior)
```

```
## Rows: 32,434,489
## Columns: 4
## $ order_id          <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3...
## $ product_id        <int> 33120, 28985, 9327, 45918, 30035, 17794, 40141, 1...
## $ add_to_cart_order <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8...
## $ reordered         <int> 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1...
```

The order_products_train contains 1.3M rows.

```
glimpse(order_products_train)
```

```
## Rows: 1,384,617
## Columns: 4
## $ order_id          <int> 1, 1, 1, 1, 1, 1, 1, 1, 36, 36, 36, 36, 36, 36, 3...
## $ product_id        <int> 49302, 11109, 10246, 49683, 43633, 13176, 47209, ...
## $ add_to_cart_order <int> 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1...
## $ reordered         <int> 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0...
```

For the kaggle competition, an order_products data set was not provided for the test orders. As a result, the order_products_train table will be partitioned into train and test for model development and model evaluation, respectively.

2. Methods/Analysis:

Data Aggregation and Partitioning The aisles and departments tables are joined to the products data table so that all product information is in one table.

```
products <- products %>% left_join(aisles) %>% left_join(departments)
```

A user_id column was added to the order_products_train. The user_ids were pulled from the orders table. Adding the user key to the table will be useful for model building as certain predictive variables related to user behavior can be created.

```
order_products_train$user_id <- orders$user_id[match(order_products_train$order_id,orders$order_id)]
```

The order_products_prior table was inner joined to the orders table. The resulting table contains all prior orders, details regarding when the orders were made, as well as, the items that were purchased. The products table was also joined to the resulting table to provide product name, department, and aisle information.

```
# Expand prior orders table in order to develop model variables
prior_set_orders_full <- orders %>% inner_join(order_products_prior,by=c("order_id"))
prior_set_orders_full <- prior_set_orders_full %>%
  left_join(select(products,product_id,product_name,aisle,department))
glimpse(prior_set_orders_full)
```

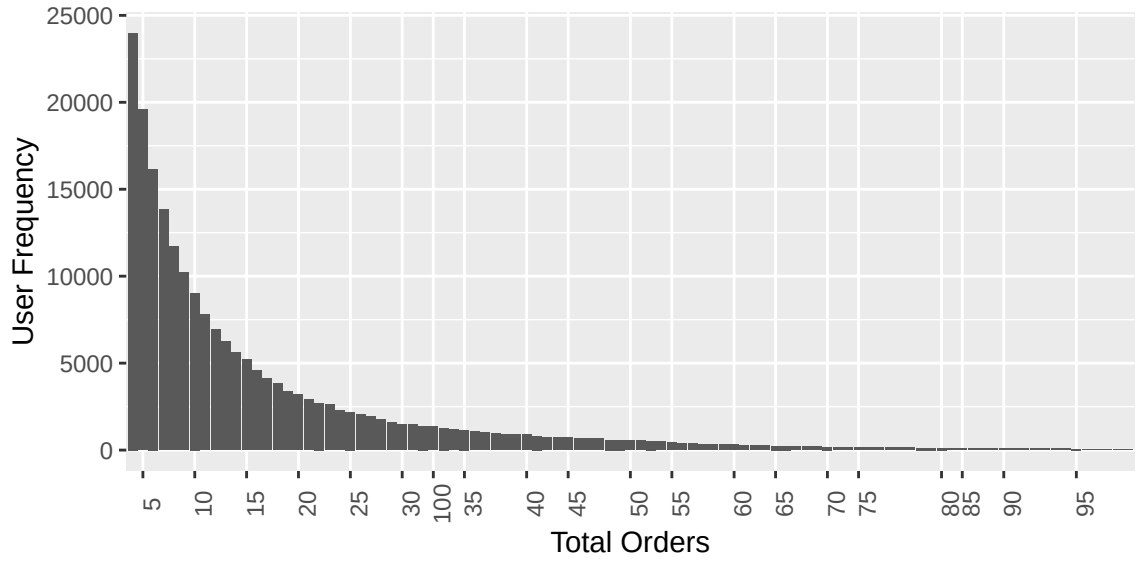
```
## Rows: 32,434,489
## Columns: 13
## $ order_id               <int> 2539329, 2539329, 2539329, 2539329, 2539329,...
## $ user_id                <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ eval_set               <fct> prior, prior, prior, prior, prior, prior, pr...
## $ order_number           <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3,...
## $ order_dow              <int> 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ order_hour_of_day      <int> 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 7, 12, 12, 12,...
## $ days_since_prior_order <dbl> NA, NA, NA, NA, NA, 15, 15, 15, 15, 15, 15, ...
## $ product_id             <int> 196, 14084, 12427, 26088, 26405, 196, 10258,...
## $ add_to_cart_order      <int> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4,...
## $ reordered              <int> 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,...
## $ product_name           <fct> Soda, Organic Unsweetened Vanilla Almond Mil...
## $ aisle                  <fct> soft drinks, soy lactosefree, popcorn jerky,...
## $ department             <fct> beverages, dairy eggs, snacks, snacks, house...
```

Ideally, the full prior_set_orders_full table would be used for model development, however due to limited computing power and time constraints, only a small subset of the table will be utilized.

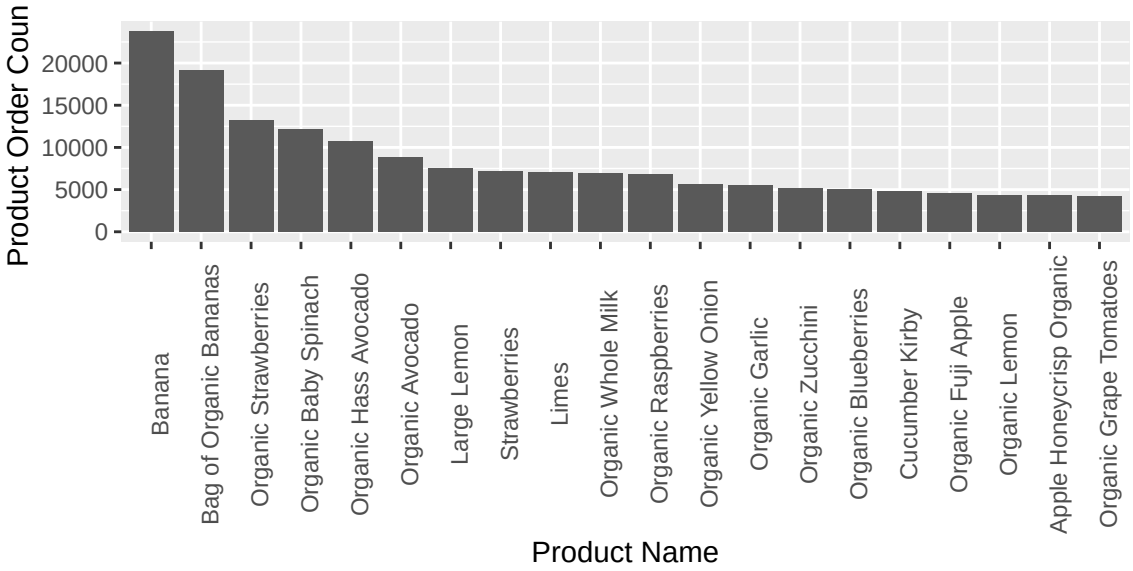
```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
subset <- createDataPartition(y = prior_set_orders_full$reordered, times = 1, p = 0.05, list = FALSE)
prior_set_orders <- prior_set_orders_full[subset,]
glimpse(prior_set_orders)
```

```
## Rows: 1,621,725
## Columns: 13
## $ order_id               <int> 2539329, 2398795, 2254736, 550135, 2295261, ...
## $ user_id                <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,...
## $ eval_set               <fct> prior, prior, prior, prior, prior, prior, pr...
## $ order_number           <int> 1, 2, 4, 7, 9, 4, 6, 6, 7, 7, 8, 9, 12, 12, ...
## $ order_dow              <int> 2, 3, 4, 1, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 4,...
## $ order_hour_of_day      <int> 8, 7, 7, 9, 16, 10, 9, 9, 12, 12, 15, 9, 9, ...
## $ days_since_prior_order <dbl> NA, 15, 29, 20, 0, 8, 13, 13, 14, 14, 27, 8,...
## $ product_id             <int> 26088, 196, 12427, 12427, 12427, 24852, 3205...
## $ add_to_cart_order      <int> 4, 1, 2, 3, 6, 8, 5, 18, 1, 5, 8, 8, 14, 19,...
## $ reordered              <int> 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,...
## $ product_name           <fct> "Aged White Cheddar Popcorn", "Soda", "Origi...
## $ aisle                  <fct> popcorn jerky, soft drinks, popcorn jerky, p...
## $ department             <fct> snacks, beverages, snacks, snacks, snacks, p...
```

Visualizations - Examining Grocery Shopping Trends in the Data Users in the dataset have placed anywhere from 4 to 100 orders, with 4 being the most frequent.



Bananas and organic bananas are the most purchased product, followed by strawberries, baby spinach, and avocado. 5 of the top 6 most purchased products are organic.



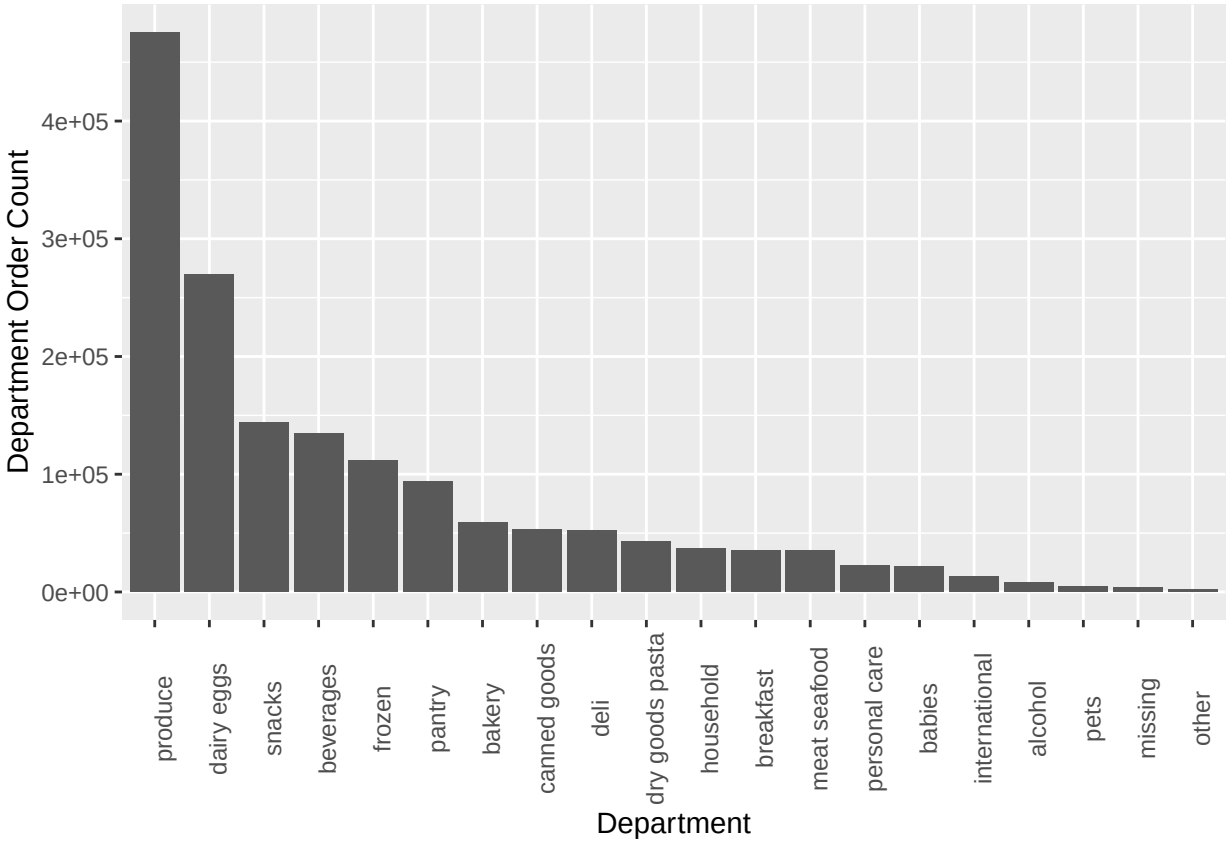
5,893 products in the data set have only been purchased once, ranging from juices, to mechanical pencils, and canola oil. Items like canola oil, mechanical pencils, and hydrocortisone cream likely take longer to finish and/or are used infrequently.

```
## # A tibble: 7,654 x 2
##    product_name                                                    count
##    <fct>                                                           <int>
##  1 "\\\"Constant Comment\\\" Green Tea Blend Tea Bags"                 1
##  2 "0% Fat Greek Yogurt Vanilla"                                       1
##  3 "0% Milkfat Greek Plain Yogurt"                                     1
##  4 "1 Ply White Luncheon Napkins"                                      1
##  5 "1/2 Caff Classic Medium Roast"                                     1
##  6 "1/2 Caff Medium Ground Coffee"                                     1
##  7 "1% Hydrocortisone Anti-Itch Liquid Maximum Strength with Healing Aloe"  1
##  8 "1% Lowfat Cultured Buttermilk"                                     1
##  9 "10 Grain Pancake & Waffle Mix"                                     1
## 10 "10 Yr Single Malt Scotch Islay"                                    1
## # ... with 7,644 more rows
```

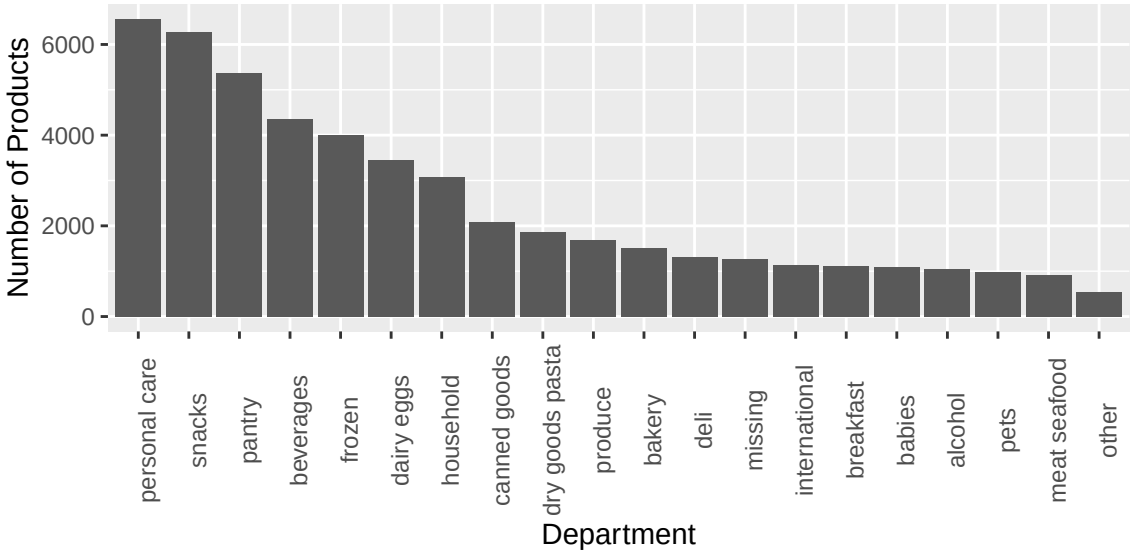
Produce, dairy eggs, and snacks are the departments from which the most products are ordered.

```
# Most Purchased Departments
prior_set_orders %>% group_by(department) %>% summarize(count=n()) %>%
  top_n(20,count) %>% arrange(desc(count)) %>% ggplot(aes(x=reorder(department,-count),y=count)) +
  geom_bar(stat="identity") + theme(axis.text.x = element_text(angle = 90)) +
  xlab("Department") + ylab("Department Order Count")
```

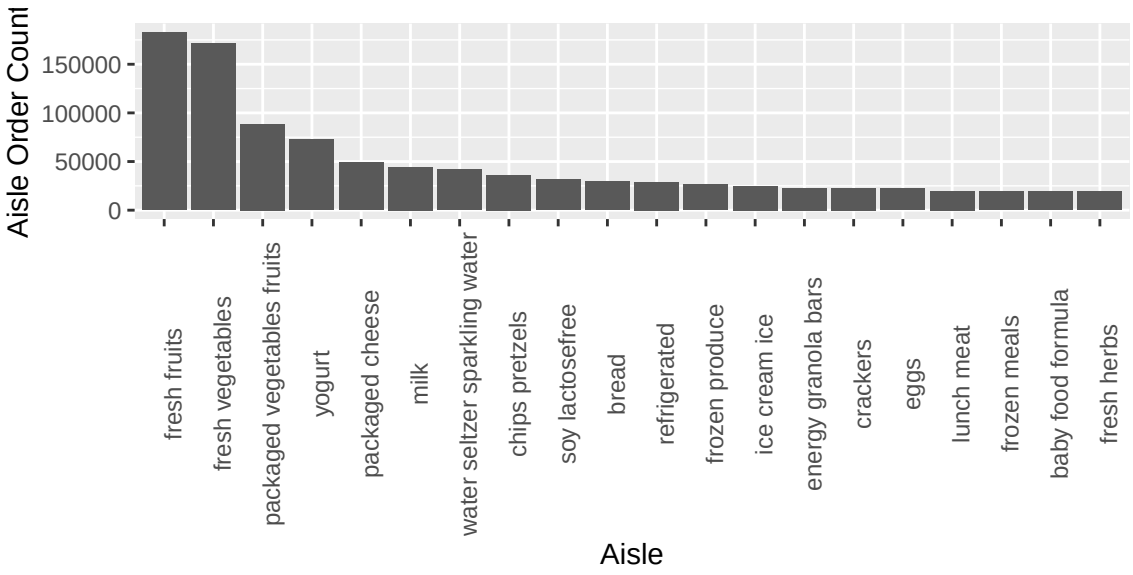
```
## `summarise()` ungrouping output (override with `.groups` argument)
```



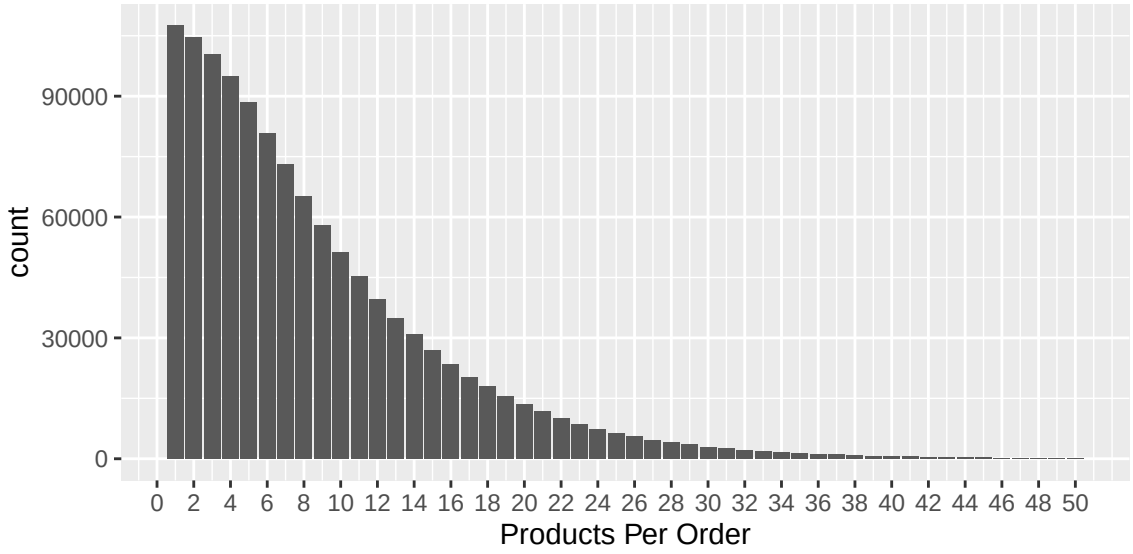
The top departments from which users buy may be skewed by the number of products within each department. For instance, 6,264 products are considered snacks, ~13% of the total products.



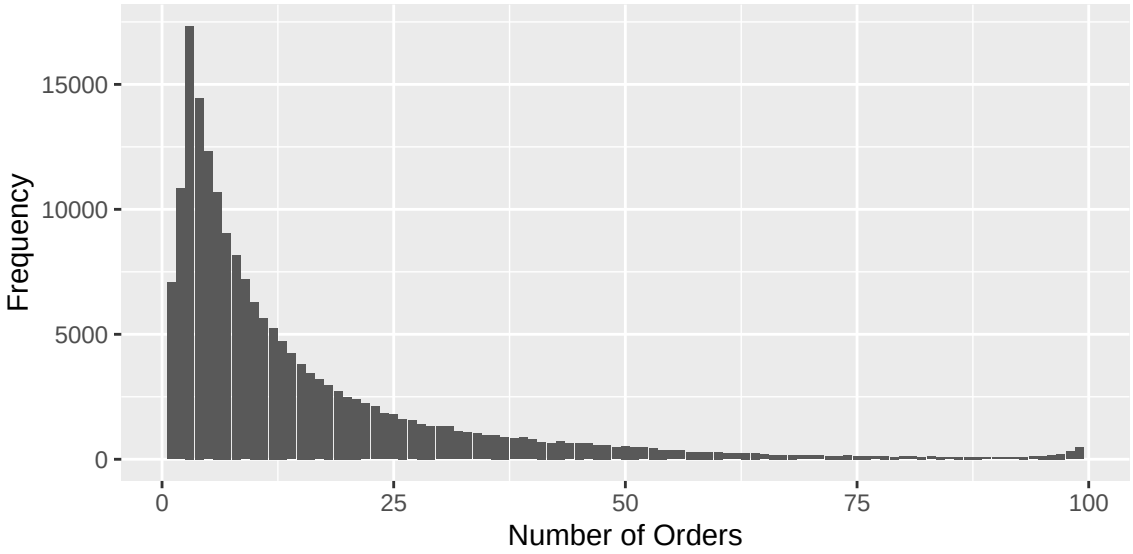
Fresh fruits, fresh vegetables are the aisles most purchased from. The top 2 aisles are nearly double the #3 aisle (packaged vegetables/fruits).



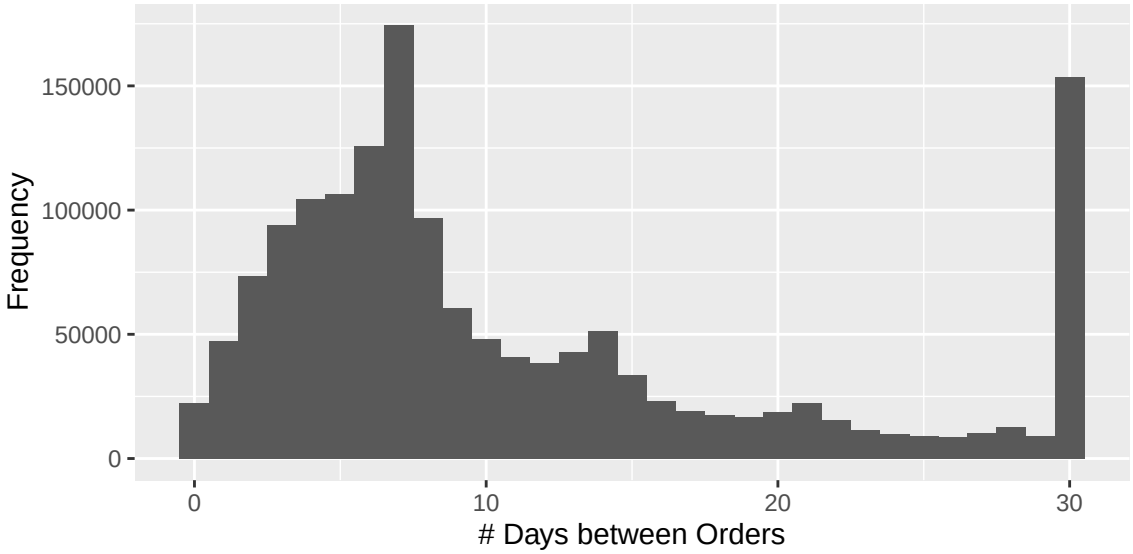
Users have purchased between 1 and 145 products per order. It is more common for users to place smaller orders, the average order includes ~9 products.



Users have made between 1 and 99 grocery orders. The median number of orders is 9.



On average users place an order every 11 days. The frequency of the 30 day bar raises some questions, for instance, if there is incentives or penalties for placing orders more or less frequently. The 30 day bar could also be a catch bucket and represent orders placed every 30 or more days. The NAs in the days_since_prior_order column would represent first time orders. There are ~208K first time orders in this data set.




```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.0     5.0     8.0    11.1    15.0    30.0  103947
```

59% of products ordered are reorders. From a preliminary examination, the percent of products that are reorders increases as the order number increases. ~27% on order 2 grows to ~51% on order 5.

```
# Reorder Frequency
sum(prior_set_orders$reordered)/length(prior_set_orders$reordered)
```

```
## [1] 0.5898645
```

```
# Number of reorders for order 2
sum(prior_set_orders$reordered[prior_set_orders$order_number == 2])/
  length(prior_set_orders$reordered[prior_set_orders$order_number == 2])
```

```
## [1] 0.2726491
```

```
# Number of reorders for order 3
sum(prior_set_orders$reordered[prior_set_orders$order_number == 3])/
  length(prior_set_orders$reordered[prior_set_orders$order_number == 3])
```

```
## [1] 0.3866474
```

```
# Number of reorders for order 4
sum(prior_set_orders$reordered[prior_set_orders$order_number == 4])/
  length(prior_set_orders$reordered[prior_set_orders$order_number == 4])
```

```
## [1] 0.4591366
```

```
# Number of reorders for order 5
sum(prior_set_orders$reordered[prior_set_orders$order_number == 5])/
  length(prior_set_orders$reordered[prior_set_orders$order_number == 5])
```

```
## [1] 0.5112043
```

Building Predictor Variables Predictor variables are necessary for the construction of the model. The original data fields are too granular so summary statistics will be calculated at the product, user and product x user levels. The summary statistics will encompass historical grocery ordering behavior which will enable us to predict future purchasing, specifically reordering probability.

Calculating Product Purchase Predictors – The variables being created are: prod_totalorders <- the total number of times a product has been purchased prod_totalreorders <- the total number of times a product has been reordered prod_totalfirstorders <- the total number of times a product has been in a user's first instacart order prod_reorderratio <- the ratio of times a product has been reordered over times a product has been purchased

```
# Product Predictors -- popularity of products
# Introducing variables specific to each product
# How often is a product purchased, reordered, ordered for the first time, reorder ratio
prod <- prior_set_orders %>% group_by(product_id) %>%
  summarize(prod_totalorders = n(),
            prod_totalreorders = sum(reordered),
            prod_totalfirstorders = sum(is.na(days_since_prior_order))) %>%
  arrange(desc(prod_totalorders))
prod <- prod %>% mutate(prod_reorderratio = prod_totalreorders/prod_totalorders)
glimpse(prod)
```

```
## Rows: 38,917
## Columns: 5
## $ product_id            <int> 24852, 13176, 21137, 21903, 47209, 47766, 476...
## $ prod_totalorders      <int> 23727, 19137, 13197, 12168, 10722, 8849, 7539...
## $ prod_totalreorders    <int> 19936, 15902, 10248, 9410, 8517, 6719, 5167, ...
## $ prod_totalfirstorders <int> 1501, 1000, 862, 744, 532, 771, 453, 425, 429...
## $ prod_reorderratio     <dbl> 0.8402242, 0.8309557, 0.7765401, 0.7733399, 0...
```

Calculating User Purchase Predictors – The variables being created are:
user_ordertotal <- the total orders a user has placed user_avgprodsperorder <- the average number of products a user has purchased per order user_distinctprodsordered <- the number of distinct products a user has purchased user_reordertotal <- the number of reorders a user has placed user_reorderprob <- the ratio of reordered items over total products purchased user_avgdaysbtworder <- the average number of days between a users orders

```
# User Predictors -- user order behavior
user <- prior_set_orders %>%
  group_by(user_id) %>%
  summarize(user_ordertotal = max(order_number),user_productordertotal = n(),
            user_avgprodsperorder = n()/max(order_number),
            user_distinctprodsordered = n_distinct(product_id),
            user_reordertotal = sum(reordered),
            user_reorderprob = sum(reordered==1)/sum(order_number>1),
            user_avgdaysbtworder = sum(days_since_prior_order,na.rm=T)/(max(order_number)-1))
glimpse(user)
```

```
## Rows: 184,918
## Columns: 8
## $ user_id                   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13...
## $ user_ordertotal           <int> 9, 14, 8, 1, 4, 1, 18, 3, 3, 5, 6, 5, 9, ...
## $ user_productordertotal    <int> 5, 12, 5, 1, 6, 1, 11, 4, 3, 5, 6, 4, 2, ...
## $ user_avgprodsperorder     <dbl> 0.5555556, 0.8571429, 0.6250000, 1.000000...
## $ user_distinctprodsordered <int> 3, 12, 3, 1, 6, 1, 11, 4, 3, 5, 6, 4, 2, ...
## $ user_reordertotal         <int> 4, 5, 3, 0, 3, 0, 7, 0, 2, 2, 2, 1, 2, 2,...
## $ user_reorderprob          <dbl> 1.0000000, 0.4166667, 0.6000000, NaN, 0.7...
## $ user_avgdaysbtworder      <dbl> 8.000000, 16.076923, 9.714286, NaN, 22.33...
```

Calculating User x Product Predictors – The variables being created are: userxprod_ordertotal <- the number of times a user has purchased a specific product userxprod_reordertotal <- the number of times a user has reordered a specific product userxprod_avg_addtocartorder <- the average order in which a user adds a product to their order

```
# User x Product Predictors -- user behavior for specific products
userxprod <- prior_set_orders %>%
  group_by(user_id,product_id) %>%
  summarize(userxprod_ordertotal = n(),
            userxprod_reordertotal = sum(reordered),
            userxprod_avg_addtocartorder = mean(add_to_cart_order))
glimpse(userxprod)
```

```
## Rows: 1,417,094
## Columns: 5
## Groups: user_id [184,918]
## $ user_id                     <int> 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,...
## $ product_id                  <int> 196, 12427, 26088, 1559, 5869, 12258, ...
## $ userxprod_ordertotal        <int> 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ userxprod_reordertotal      <int> 1, 3, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,...
## $ userxprod_avg_addtocartorder <dbl> 1.000000, 3.666667, 4.000000, 3.000000...
```

The product predictors, user predictors, and product x user predictors were aggregated into a single table. The predictors are joined to the order_products_train set to create the order_pred table. The order_pred

table contains the user_id and product_id for a subset of the latest instacart orders. The order_pred table will be used to develop models predicting if a user will reorder specific grocery items.

```
vars <- userxprod %>% inner_join(prod) %>% inner_join(user)
order_pred_set <- vars %>% left_join(select(order_products_train,user_id,
                                            product_id,reordered))
order_pred_set$reordered[is.na(order_pred_set$reordered)] <- 0
```

The order_pred table must be partitioned into train, test, and validation datasets. The models will be developed using train. Model parameters and preliminary model evaluation will be done against the test set. Validation will be used to evaluate the final model. Partitioning the data ensures that created models are not overfit to the training dataset. The full data set was subset to 200K to account for available computing power.

```
set.seed(1, sample.kind="Rounding")
order_pred <- order_pred_set[1:200000,]

set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y=order_pred$reordered,times=2,p=0.2,list = FALSE)
train <- order_pred[-test_index[,1],]
test <- order_pred[test_index[,1],]
validate <- order_pred[test_index[,2],]
```

3. Results: (presents modeling results and discusses model performance)

Using XGBoost to select most predictive variables. The XGBoost package will be utilized to determine what variables are most important for variable prediction. In prior steps, a number of predictors were constructed but all variables may not be predictive and there may be correlations between variables. For the purposes of this analysis, the XGBoost will run to on a logistic regression model and evaluated using the logloss metric. Accuracy would not be appropriate for the instacart data due to the nature of the reordered variable. In the train set, ~10% of product orders are reordered products. Therefore a prediction of 100% reorders would result in 100% accuracy. Logloss is best used to measure the probability of a binary response variable. Logloss is a more appropriate performance metric because there is a penalty for incorrect classification.

```
params <- list("objective"= "reg:logistic","eval_metric"="logloss")
X <- xgb.DMatrix(as.matrix(train %>% select(-reordered)), label = train$reordered)
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
logmodel_vartest <- xgboost(data = X, params = params, nrounds = 80)
```

```
## [1]  train-logloss:0.536127
## [2]  train-logloss:0.451924
## [3]  train-logloss:0.402508
## [4]  train-logloss:0.372277
## [5]  train-logloss:0.353439
## [6]  train-logloss:0.341532
## [7]  train-logloss:0.334032
```

```
## [8]  train-logloss:0.329109
## [9]  train-logloss:0.325901
## [10] train-logloss:0.323651
## [11] train-logloss:0.322063
## [12] train-logloss:0.320742
## [13] train-logloss:0.319640
## [14] train-logloss:0.318768
## [15] train-logloss:0.318117
## [16] train-logloss:0.317100
## [17] train-logloss:0.316440
## [18] train-logloss:0.315676
## [19] train-logloss:0.315208
## [20] train-logloss:0.314666
## [21] train-logloss:0.314143
## [22] train-logloss:0.313578
## [23] train-logloss:0.313228
## [24] train-logloss:0.312698
## [25] train-logloss:0.312321
## [26] train-logloss:0.311819
## [27] train-logloss:0.311535
## [28] train-logloss:0.310946
## [29] train-logloss:0.310370
## [30] train-logloss:0.310040
## [31] train-logloss:0.308713
## [32] train-logloss:0.308087
## [33] train-logloss:0.307847
## [34] train-logloss:0.307675
## [35] train-logloss:0.307330
## [36] train-logloss:0.307162
## [37] train-logloss:0.306950
## [38] train-logloss:0.306657
## [39] train-logloss:0.306268
## [40] train-logloss:0.305340
## [41] train-logloss:0.304795
## [42] train-logloss:0.304529
## [43] train-logloss:0.304020
## [44] train-logloss:0.303620
## [45] train-logloss:0.303099
## [46] train-logloss:0.302755
## [47] train-logloss:0.302409
## [48] train-logloss:0.302016
## [49] train-logloss:0.301631
## [50] train-logloss:0.300955
## [51] train-logloss:0.300525
## [52] train-logloss:0.299885
## [53] train-logloss:0.299586
## [54] train-logloss:0.299357
## [55] train-logloss:0.298826
## [56] train-logloss:0.298651
## [57] train-logloss:0.298129
## [58] train-logloss:0.297483
## [59] train-logloss:0.297355
## [60] train-logloss:0.296909
## [61] train-logloss:0.296748
```

```
## [62] train-logloss:0.296168
## [63] train-logloss:0.295629
## [64] train-logloss:0.295505
## [65] train-logloss:0.295207
## [66] train-logloss:0.295030
## [67] train-logloss:0.294702
## [68] train-logloss:0.294214
## [69] train-logloss:0.293424
## [70] train-logloss:0.292738
## [71] train-logloss:0.292218
## [72] train-logloss:0.291847
## [73] train-logloss:0.291300
## [74] train-logloss:0.290622
## [75] train-logloss:0.290248
## [76] train-logloss:0.289851
## [77] train-logloss:0.289308
## [78] train-logloss:0.288965
## [79] train-logloss:0.288550
## [80] train-logloss:0.288154
```

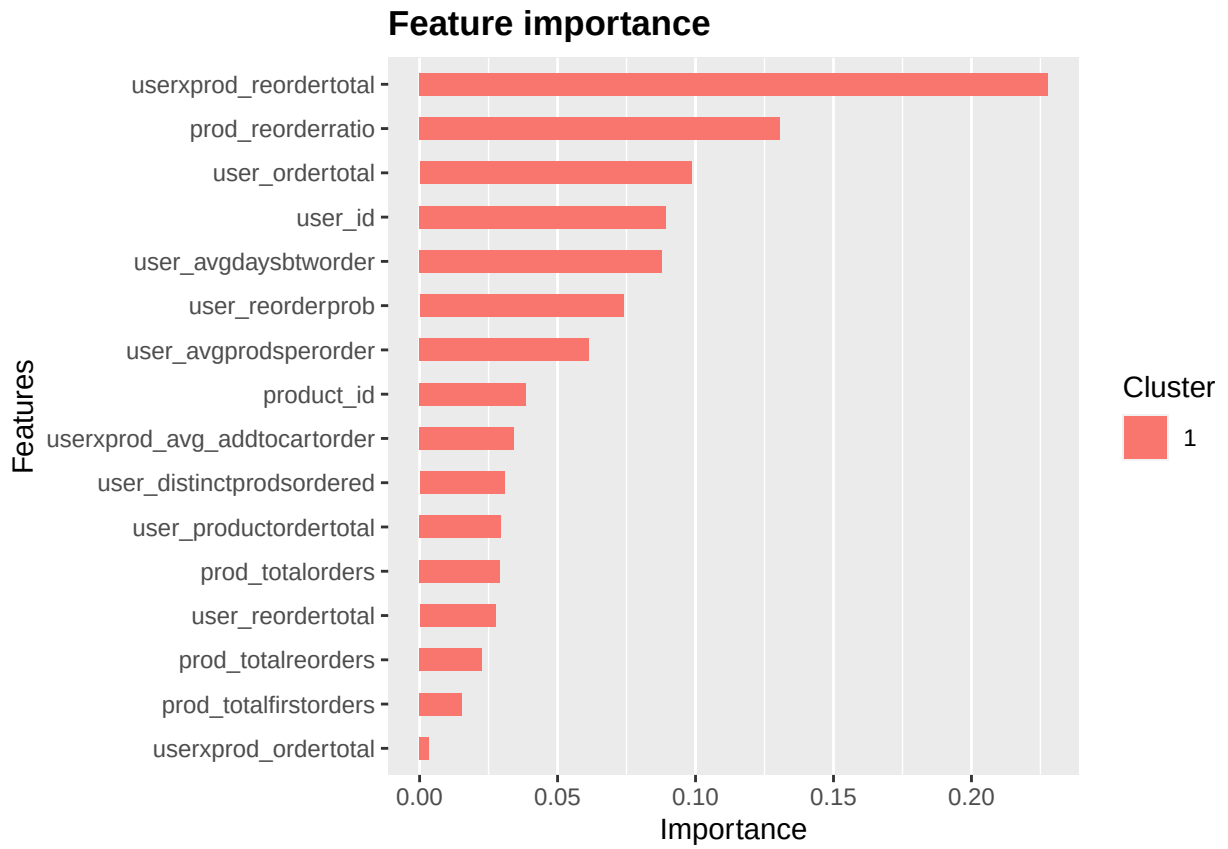
XGBoost logistic regression ran 80 model iterations, testing different predictor variable combinations to optimize logloss. As the models are iterated through, variables are evaluated on 4 metrics: gain, cover, frequency, and importance.

```
# We estimate the importance of the predictors
var_importance <- xgb.importance(colnames(X), model = logmodel_vartest)
var_importance
```

```
##                          Feature        Gain       Cover   Frequency
##  1:       userxprod_reordertotal 0.227608476 0.061619534 0.018346253
##  2:            prod_reorderratio 0.130702820 0.104643293 0.095348837
##  3:              user_ordertotal 0.098562305 0.090095264 0.075968992
##  4:                      user_id 0.089221158 0.104434937 0.135658915
##  5:         user_avgdaysbtworder 0.087776119 0.126389804 0.117312661
##  6:             user_reorderprob 0.073931413 0.074550934 0.095865633
##  7:        user_avgprodsperorder 0.061517847 0.100156162 0.083720930
##  8:                   product_id 0.038481108 0.078291110 0.082687339
##  9: userxprod_avg_addtocartorder 0.034201393 0.045447140 0.059431525
## 10:    user_distinctprodsordered 0.030731881 0.057268084 0.043152455
## 11:       user_productordertotal 0.029434832 0.048116708 0.040826873
## 12:            prod_totalorders 0.029060315 0.031029423 0.050645995
## 13:            user_reordertotal 0.027541759 0.044481256 0.035917313
## 14:           prod_totalreorders 0.022595317 0.019391121 0.031782946
## 15:        prod_totalfirstorders 0.015370217 0.008964145 0.028423773
## 16:         userxprod_ordertotal 0.003263041 0.005121087 0.004909561
```

The ensembling results indicate that the number of times a user reorders a specific product, is the most predictive variable. The number of times a product is reordered is the second most important feature. To start modeling, the top 7 features will be included (greater than 0.05 importance).

```
# Plot importance by predictor
xgb.ggplot.importance(var_importance)
```



Model 1: Logistic Regression using selected variables With the selected variables, the first model built is a logistic regression model. Using the predict function, the logistic regression model will produce a probability of reorder. The predictions are converted to a binary result using a 0.5 cutoff. Any probability less than or equal to 0.5 is set as not reordered, any probability greater than 0.5 is set as reordered.

```
# Build the logistic regression model using the train data set
log_fit <- lm(reordered ~ userxprod_reordertotal +
                prod_reorderratio + user_ordertotal +
                user_id + user_avgdaysbtworder + user_reorderprob +
                user_avgprodsperorder, data=train)
log_fit
```

```
##
## Call:
## lm(formula = reordered ~ userxprod_reordertotal + prod_reorderratio +
##     user_ordertotal + user_id + user_avgdaysbtworder + user_reorderprob +
##     user_avgprodsperorder, data = train)
##
## Coefficients:
##            (Intercept)  userxprod_reordertotal       prod_reorderratio
##              -5.960e-02               7.114e-02               1.899e-01
##        user_ordertotal                 user_id    user_avgdaysbtworder
##              -1.442e-03               4.582e-08               9.659e-04
##       user_reorderprob   user_avgprodsperorder
##               8.284e-02               9.995e-04
```

```r
# p_hat probability of reorder
p_hat <- predict(log_fit,test)
summary(p_hat)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
## -0.15600  0.06195  0.11428  0.11554  0.16892  0.57521      221
```

```r
y_hat <- factor(ifelse(p_hat > 0.5,1,0))
length(y_hat[y_hat==1])/length(y_hat)
```

```
## [1] 0.0058
```

```r
# 0.94
F_meas(y_hat,factor(test$reordered))
```

```
## [1] 0.9382133
```

```r
# Confusion Matrix
confusionMatrix(y_hat,factor(test$reordered))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 35145  4623
##          1     6     5
##
##                Accuracy : 0.8836
##                  95% CI : (0.8804, 0.8868)
##     No Information Rate : 0.8837
##     P-Value [Acc > NIR] : 0.5102
##
##                   Kappa : 0.0016
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99983
##             Specificity : 0.00108
##          Pos Pred Value : 0.88375
##          Neg Pred Value : 0.45455
##              Prevalence : 0.88366
##          Detection Rate : 0.88351
##    Detection Prevalence : 0.99972
##       Balanced Accuracy : 0.50045
##
##        'Positive' Class : 0
##
```

The resulting F1 score for the logistic regression is 0.94. The F1 score indicates a well performing model, however the confusion matrix illustrates that the model predicts a reorder rate of 0.03% when the actual

reorder rate is 11.6%. The balanced accuracy is 0.50. The model is conservative and is skewed to predicting not reordered.

Model 2: Random Forest The second model used to fit the data is a random forest model. The random forest model averages multiple decision trees using bootstrapping. The random forest model is an improvement to the logistic regression model. The F1 score is less than the logistic regression 0.94 to 0.92, however the specificity – correctly predicting not reordered was greatly improved. Specificity went from 0.001 to 0.13.

```
train$reordered <- as.factor(train$reordered)
rf <- randomForest(reordered ~ userxprod_reordertotal +
                     prod_reorderratio + user_ordertotal +
                     user_id + user_avgdaysbtworder + user_reorderprob +
                     user_avgprodsperorder, method='rpart',
                   data = train, ntree = 5, na.action = na.exclude)
y_hat_rf <- predict(rf,test)
# 0.93
F_meas(y_hat_rf,factor(test$reordered))
```

```
## [1] 0.9263701
```

```
confusionMatrix(y_hat_rf,factor(test$reordered))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 33756  3971
##          1  1395   657
##
##                Accuracy : 0.8651
##                  95% CI : (0.8617, 0.8684)
##     No Information Rate : 0.8837
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1349
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9603
##             Specificity : 0.1420
##          Pos Pred Value : 0.8947
##          Neg Pred Value : 0.3202
##              Prevalence : 0.8837
##          Detection Rate : 0.8486
##    Detection Prevalence : 0.9484
##       Balanced Accuracy : 0.5511
##
##        'Positive' Class : 0
##
```

I ran a random forest, increasing the number of trees averaged from 5 to 50 to compare the results. There was some improvement as the positive predictive value went from 0.88 to 0.89.

```
train$reordered <- as.factor(train$reordered)
rf <- randomForest(reordered ~ userxprod_reordertotal +
                     prod_reorderratio + user_ordertotal +
                     user_id + user_avgdaysbtworder + user_reorderprob +
                     user_avgprodsperorder, method='rpart',
                   data = train, ntree = 50, na.action = na.exclude)
y_hat_rf <- predict(rf,test)
# 0.93
F_meas(y_hat_rf,factor(test$reordered))
```

```
## [1] 0.9350231
```

```
confusionMatrix(y_hat_rf,factor(test$reordered))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 34565  4218
##          1   586   410
##
##                Accuracy : 0.8792
##                  95% CI : (0.876, 0.8824)
##     No Information Rate : 0.8837
##     P-Value [Acc > NIR] : 0.997
##
##                   Kappa : 0.1091
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98333
##             Specificity : 0.08859
##          Pos Pred Value : 0.89124
##          Neg Pred Value : 0.41165
##              Prevalence : 0.88366
##          Detection Rate : 0.86893
##    Detection Prevalence : 0.97496
##       Balanced Accuracy : 0.53596
##
##        'Positive' Class : 0
##
```

4. Conclusion: (brief summary of report, limitations, future work)

In modeling both the logistic regression and the random forest models, the random forest approach was better able to predict reorders. The random forest produced a more balanced model. The logistic regression had a high sensitivity but a very low specificity. The data set is particularly difficult because the response variable is heavily skewed, only ~11% of the orders are reorders. Although the kaggle data set contains a lot of data (~32M rows), the report is limited in its ability to evaluate the full data set do to the computing constraints. To continue this analysis, there are additional variables that would be investigated. For instance, the increase in reorder rate as the order number increases. Additionally, department and aisle were not included as predictive variables but could provide additional insight. In evaluating others' approaches to the

instacart data there are a lot of different models and packages that would be useful to enhance the machine learning. The edX course provided a great foundational knowledge of data science. I hope to continue to expand my exposure and understanding of the different modeling capabilities and continue to enhance my data science knowledge.