Northeastern University
College of Professional Studies

# Final Project: Market Basket Analysis on Instacart orders for 2017

Dickson Wanjau

ALY 6040 81061 Data Mining Applications
SPRING 2019 CPS

Instructor: Dr. Amin Karimpour

Due Date: June 11, 2019.

# Introduction

Our team will be conducting a Market Basket Analysis on an open source dataset in Kaggle web link: https://www.instacart.com/datasets/grocery-shopping-2017 . The dataset contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. The goal of the competition is to predict which products will be in a user's next order.

Market Basket Analysis is one of the key techniques used by large retailers to uncover associations between items purchased. It allows retailers to identify relationships between the items that people buy. It can tell them what items customers frequently buy together by generating a set of rules called Association Rules. It gives an output as rules in form “if this then that.” The result of a market basket analysis is a collection of association rules that specify patterns found in the relationships among items in the itemset. Association Rules are used to find association between different objects in the itemset.

Groups of one or more items that in a given transaction are surrounded by curly brackets to indicate an itemset. For example, let’s consider the following purchases made at a store: {Bread, Milk}, {Bread, Milk, Sugar, Eggs}, {Bread, Milk, Cigarettes, Candy}, {Eggs, Milk, Butter}, {Bread, Milk, Toothpaste, Cheese} etc.

From the above transactions, we can observe that bread is bought with milk in 4 transactions. Similarly, eggs are bought with milk in 3 transactions making them both frequent item sets.

Association rules are always composed from subsets of itemset and are denoted by relating one itemset on the left-hand side (LHS) of the rule to another itemset on the right-hand side (RHS) of the rule. The LHS is the condition that needs to be met in order to trigger the rule, and the RHS is the expected result of meeting that condition. Association rules are used for unsupervised knowledge discovery in large databases not prediction.

Association Rules are given in the form below;

{A, B, C, D} $\rightarrow$ {E}        read as, if A (Antecedent) then B(Consequent).

Support and Confidence measure how interesting the rule is. It is set by the minimum support and minimum confidence thresholds.

For example: $\{\text{Bread, Eggs, Sugar}\} \rightarrow \{\text{Milk}\}$

This association rule states that if bread, eggs and sugar are purchased together, then milk is also likely to be purchased. In other words, "bread eggs and sugar imply bread."

Retailers can use those rules for numerous marketing strategies such as:

- Changing the store layout according to trends

- Customer behavior analysis
- Customize catalogue design
- Identifying what are the trending items customers buy
- Customizing emails with add-on sales

We will use the Apriori Algorithm in R to conduct our Market Basket Analysis.

Typically, transactional datasets are typically extremely large, both in terms of the number of transactions as well as the number of items or features that are monitored. The problem is that the number of potential item sets grows exponentially with the number of features.

The Apriori algorithm tries to reduce this learning space by reducing the items sets combinations that are not frequent. It employs a simple a prior (hence the name 'A priori') belief to reduce the association rule search space: all subsets of a frequent itemset must also be frequent. This is known as the Apriori property. For example, the set {cooking oil, lipstick} can only be frequent if both {cooking oil} and {lipstick} occur frequently as well.

In practice, a set of {cooking oil, lipstick} is likely to be uncommon if it ever occurs. By ignoring these rare (and, perhaps, less important) combinations, it is possible to limit the scope of the search for rules to a more manageable size.

## Measuring association rule interest – support and confidence

Let's again consider our prior transactions.

| **Transaction Number** | **Items Purchased** |
|---|---|
| 1 | {Bread, Milk}, |
| 2 | {Bread, Milk, Sugar, Eggs} |
| 3 | {Bread, Milk, Cigarettes, Candy} |
| 4 | {Eggs, Milk, Butter, Cheese}, |

| | |
|---|---|
| 5 | {Bread, Milk, Toothpaste, Cheese} |

The support of an itemset or rule measures how frequently it occurs in the data. Specifically, we define support as: $Support(X) = \frac{count(X)}{N}$ where count(X) number of transactions containing itemset X. N is the number of transactions in the database.

For instance, the itemset {Bread, Milk}, has support of 4 / 5 = 0.8 in the transactional data. The support can be calculated for any itemset or even a single item; for example, the support for {Cheese} is 2 / 5 = 0.4.

A rule's confidence is a measurement of its predictive power or accuracy. It is defined as the support of the itemset containing both X and Y divided by the support of the itemset containing only X: $Confidence(X \rightarrow Y) = \frac{Support(X,Y)}{Support(X)}$

Confidence tells us the proportion of transactions where the presence of item or itemset X results in the presence of item or itemset Y.

For example, the confidence of {Bread} → {Milk} is 0.8 / 1.0 = 0.80.

This means that a purchase involving bread is accompanied by a purchase of milk 80 percent of the time.

# Data Analysis I

## R Libraries Used

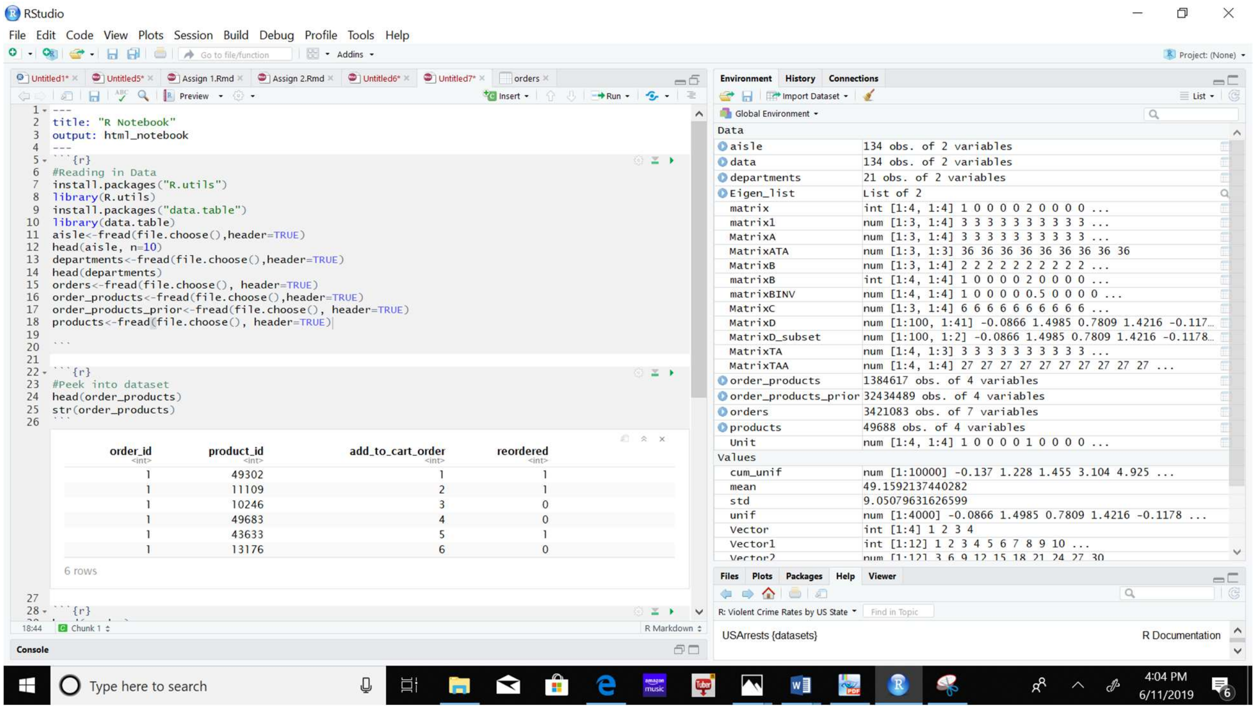
The following are the requisite libraries that we need in our market basket analysis:

| Package | Description |
|---|---|
| arules | Provides the infrastructure for representing, manipulating and analyzing transaction data and patterns. |
| arulesViz | Extends the capabilities of package 'arules' with various visualization techniques for association rules and item-sets. |
| tidyverse | Includes the requisite packages used for most data analysis like ggplot and dplyr. |
| data.table | Helps in fast file reading into R |

First, we load the necessary libraries and read in the data into R.

```
> library(data.table)
> library(R.utils)
> orders<-fread(file.choose(), header=TRUE)
> aisle<-fread(file.choose(),header=TRUE)
> departments<-fread(file.choose(),header=TRUE)
> order_products<-fread(file.choose(),header=TRUE)
> order_products_prior<-fread(file.choose(), header=TRUE)
> products<-fread(file.choose(), header=TRUE)
```

The below screenshot shows the initial installation and importation of the data into R.




# Peek into dataset

```
> head(orders)
```

| order_id <int> | user_id <int> | eval_set <chr> | order_number <int> | order_dow <int> | order_hour_of_day <int> |
|---|---|---|---|---|---|
| 2539329 | 1 | prior | 1 | 2 | 8 |
| 2398795 | 1 | prior | 2 | 3 | 7 |
| 473747 | 1 | prior | 3 | 3 | 12 |
| 2254736 | 1 | prior | 4 | 4 | 7 |
| 431534 | 1 | prior | 5 | 4 | 15 |
| 3367565 | 1 | prior | 6 | 2 | 7 |

6 rows | 1-6 of 7 columns

| user_id <int> | eval_set <chr> | order_number <int> | order_dow <int> | order_hour_of_day <int> | days_since_prior_order <dbl> |
|---|---|---|---|---|---|
| 1 | prior | 1 | 2 | 8 | NA |
| 1 | prior | 2 | 3 | 7 | 15 |
| 1 | prior | 3 | 3 | 12 | 21 |
| 1 | prior | 4 | 4 | 7 | 29 |
| 1 | prior | 5 | 4 | 15 | 28 |
| 1 | prior | 6 | 2 | 7 | 19 |

```
> str(orders)
Classes ‘data.table’ and 'data.frame':3421083 obs. of  7 variables:
 $ order_id              : int  2539329 2398795 473747 2254736 431534 3367565
550135 3108588 2295261 2550362 ...
 $ user_id               : int  1 1 1 1 1 1 1 1 1 1 ...
 $ eval_set              : chr  "prior" "prior" "prior" "prior" ...
 $ order_number          : int  1 2 3 4 5 6 7 8 9 10 ...
 $ order_dow             : int  2 3 3 4 4 2 1 1 1 4 ...
 $ order_hour_of_day     : int  8 7 12 7 15 7 9 14 16 8 ...
```

```
 $ days_since_prior_order: num  NA 15 21 29 28 19 20 14 0 30 ...
```

As observed, the "orders" table, contains 3,421,083 records of orders made by customers. It contains 7 features of different data types. For example, the user_id is of integer data type.

This csv file gives a list of all orders we have in the dataset. 1 row per order. For example, we can see that user 1 has 11 orders, 1 of which is in the train set, and 10 of which are prior orders. However, it doesn't tell us about which products were ordered. This is contained in the order_products.csv shown below.

```
> head(order_products)
```

| order_id <int> | product_id <int> | add_to_cart_order <int> | reordered <int> |
|---|---|---|---|
| 1 | 49302 | 1 | 1 |
| 1 | 11109 | 2 | 1 |
| 1 | 10246 | 3 | 0 |
| 1 | 49683 | 4 | 0 |
| 1 | 43633 | 5 | 1 |
| 1 | 13176 | 6 | 0 |

6 rows

This file gives us information about which products (product_id) were ordered. It also contains information of the order (add_to_cart_order) in which the products were put into the cart and information of whether this product is a re-order(1) or not(0).

For example, we see below that order_id 1 had 6 products, 3 of which are reorders.

We look at the products csv file which specifies what these products are.

```
> head(products)
```

| product_id <int> | product_name <chr> |
|---|---|
| 1 | Chocolate Sandwich Cookies |
| 2 | All-Seasons Salt |
| 3 | Robust Golden Unsweetened Oolong Tea |
| 4 | Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce |
| 5 | Green Chile Anytime Sauce |
| 6 | Dry Nose Oil |

6 rows | 1-2 of 4 columns

| aisle_id <int> | department_id <int> |
|---|---|
| 61 | 19 |
| 104 | 13 |
| 94 | 7 |
| 38 | 1 |
| 5 | 13 |
| 11 | 11 |

6 rows | 3-4 of 4 columns

It shows the products with their corresponding product_id. Further, it shows the aisle and department ids.

```
> str(products)
Classes ‘data.table’ and 'data.frame':49688 obs. of  4 variables:
 $ product_id   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ product_name : chr  "Chocolate Sandwich Cookies" "All-Seasons Salt" "Robus
t Golden Unsweetened Oolong Tea" "Smart Ones Classic Favorites Mini Rigatoni
With Vodka Cream Sauce" ...
 $ aisle_id     : int  61 104 94 38 5 11 98 116 120 115 ...
 $ department_id: int  19 13 7 1 13 11 7 1 16 7 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

There are 49,688 products in the catalogue within 134 aisles and 21 departments.

The aisle csv file shows the different aisles in the store as follows:

```
> head(aisle)
```

| aisle_id <int> | aisle <chr> |
|---|---|
| 1 | prepared soups salads |
| 2 | specialty cheeses |
| 3 | energy granola bars |
| 4 | instant foods |
| 5 | marinades meat preparation |
| 6 | other |

6 rows

There are 134 aisles in our dataset. Here’s a breakdown of a few

```
> str(aisle)
Classes ‘data.table’ and 'data.frame':134 obs. of  2 variables:
 $ aisle_id: int  1 2 3 4 5 6 7 8 9 10 ...
 $ aisle   : chr  "prepared soups salads" "specialty cheeses" "energy granola
bars" "instant foods" ...
 - attr(*, ".internal.selfref")=<externalptr>
```

```
> paste(sort(head(aisle$aisle)), collapse=', ')
[1] "energy granola bars, instant foods, marinades meat preparation, other, p
repared soups salads, specialty cheeses"
```

The deaprtments csv file shows the various departments in the store.

```
> head(departments, n=10)
```

| department_id <int> | department <chr> |
|---|---|
| 1 | frozen |
| 2 | other |
| 3 | bakery |
| 4 | produce |
| 5 | alcohol |
| 6 | international |
| 7 | beverages |
| 8 | pets |
| 9 | dry goods pasta |
| 10 | bulk |

1-10 of 10 rows

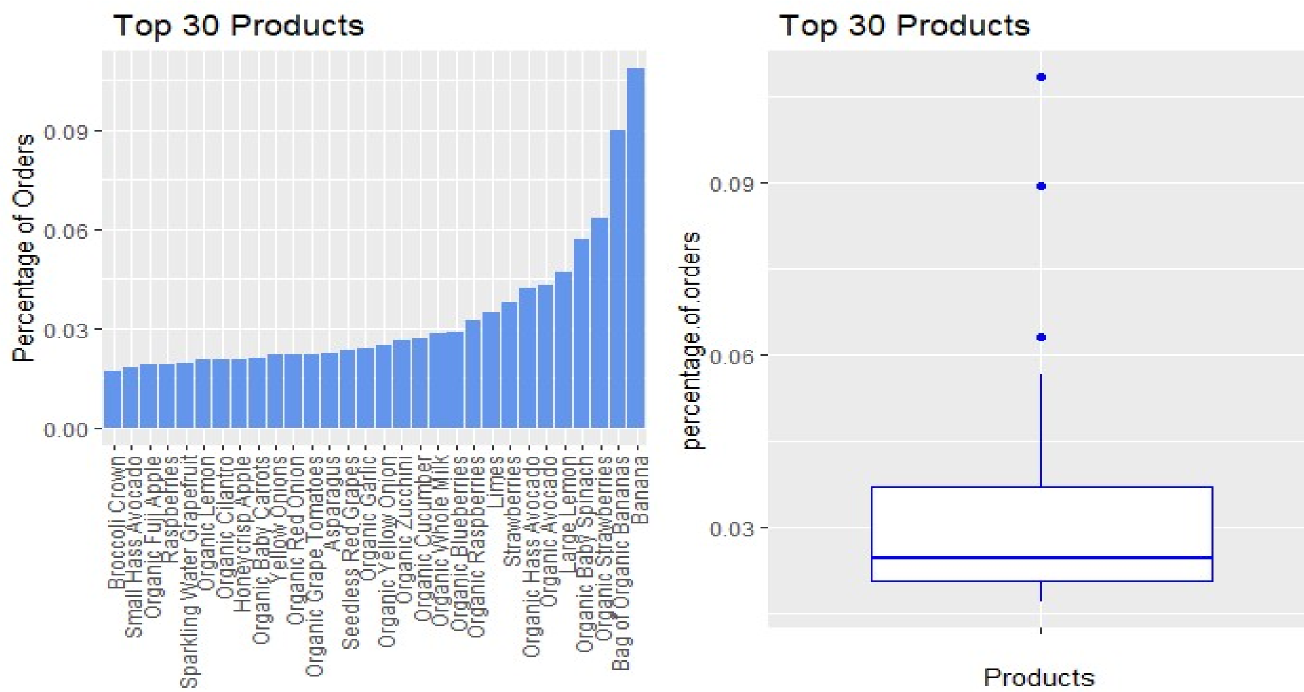
There are 21 departments in this dataset. The names of all departments are listed below.

```
> str(departments)
Classes ‘data.table’ and 'data.frame':21 obs. of  2 variables:
 $ department_id: int  1 2 3 4 5 6 7 8 9 10 ...
 $ department   : chr  "frozen" "other" "bakery" "produce" ...
 - attr(*, ".internal.selfref")=<externalptr>
> paste(sort(departments$department), collapse = ', ')
[1] "alcohol, babies, bakery, beverages, breakfast, bulk, canned goods, dairy
eggs, deli, dry goods pasta, frozen, household, international, meat seafood,
missing, other, pantry, personal care, pets, produce, snacks"
```

# Exploratory Analysis

## Most Popular Products Sold

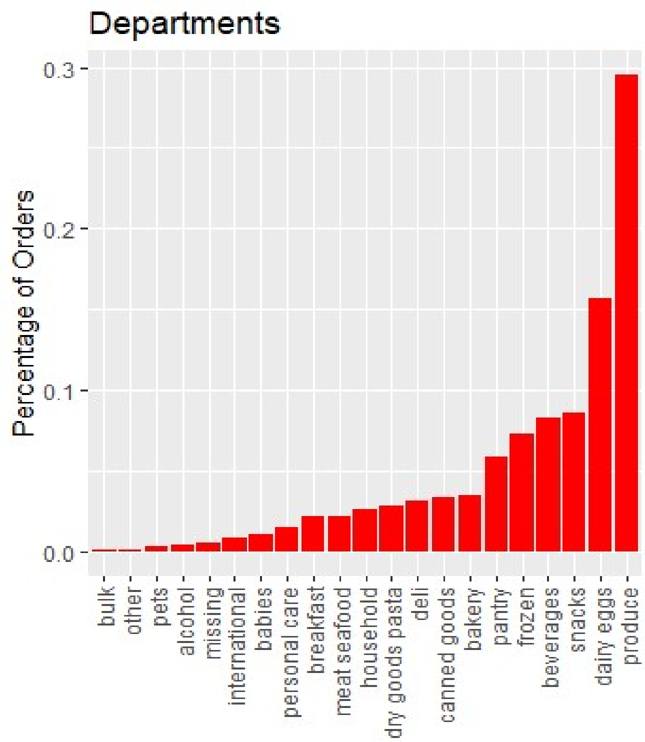
```
> library(tidyverse)
> tmp = order_products %>%
+   left_join(products) %>%
+   group_by(product_name) %>%
+   summarize(count=n()) %>%
+   top_n(n=30, wt=count) %>%  mutate(percentage=count/sum(count))
> p1 = ggplot (tmp, aes(x=reorder(product_name,count), y=percentage)) +
+   geom_col(fill="cornflowerblue") + ggtitle('Products Top 30') + ylab('Perc
entage of Orders') +
+   theme (
+     axis.text.x=element_text(angle=90, hjust=1, vjust=0.5),
+     axis.title.x = element_blank())
> p2 = ggplot (data = tmp, aes( x= '', y=percentage )) +
+   ggtitle(' Top 30 Products') + ylab('percentage.of.orders') + geom_boxplot
(color="blue") + xlab('Products')
> grid.arrange(p1, p2, ncol = 2)
```

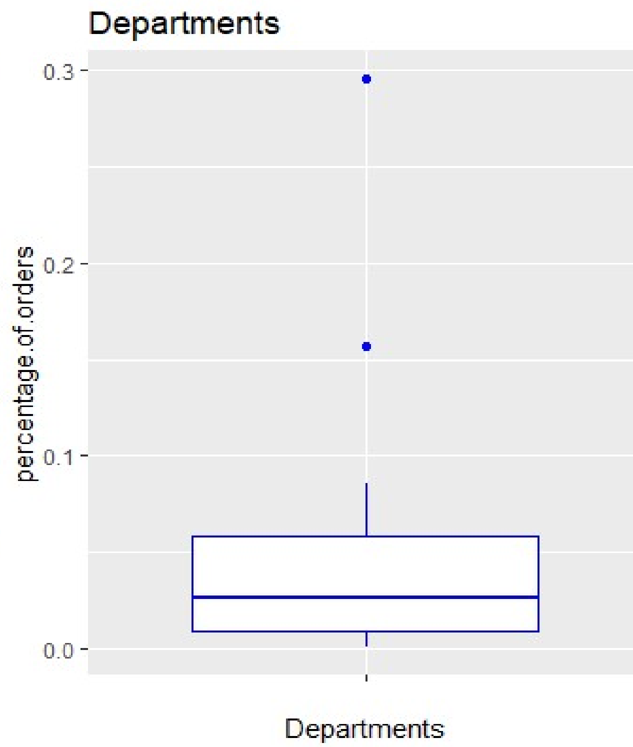



Banana are the most popular products. The number of orders varies greatly for different products. The illustration above shows sample of only 30 top products.

## Most Popular Department Sold

```
> grid.arrange(p1, p2, ncol = 2)
> tmp = order_products %>%
+   left_join(products) %>%
+   left_join(departments) %>%
+   group_by(department) %>%
+   summarize(count=n()) %>%
+   mutate(percentage=count/sum(count))
> p1 = ggplot (tmp, aes(x=reorder(department,count), y=percentage)) +
+   geom_col(fill="red") + ggtitle('Departments') + ylab('Percentage of Order
s') +
+   theme (
+     axis.text.x=element_text(angle=90, hjust=1, vjust=0.5),
+     axis.title.x = element_blank())
> p2 = ggplot (data = tmp, aes( x= '', y=percentage )) +
+   ggtitle('Departments') + ylab('percentage.of.orders') + geom_boxplot(colo
r="blue") + xlab('Departments')
> grid.arrange(p1, p2, ncol = 2)
```



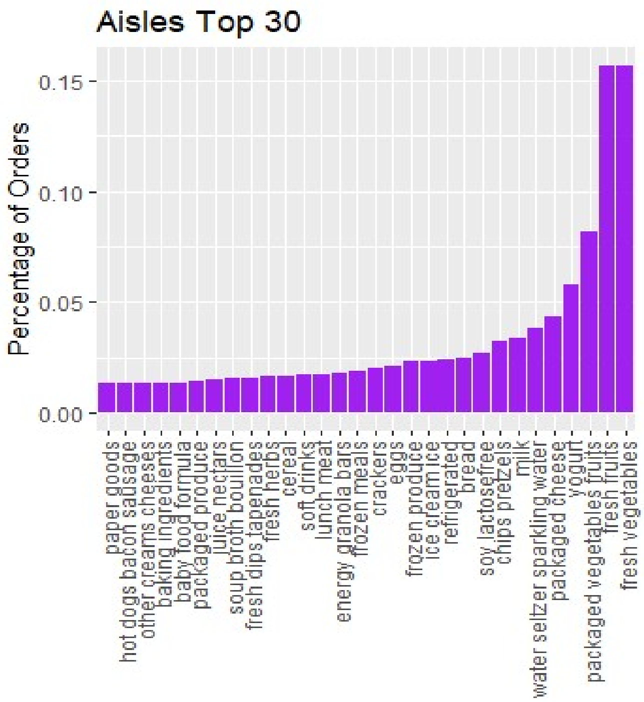



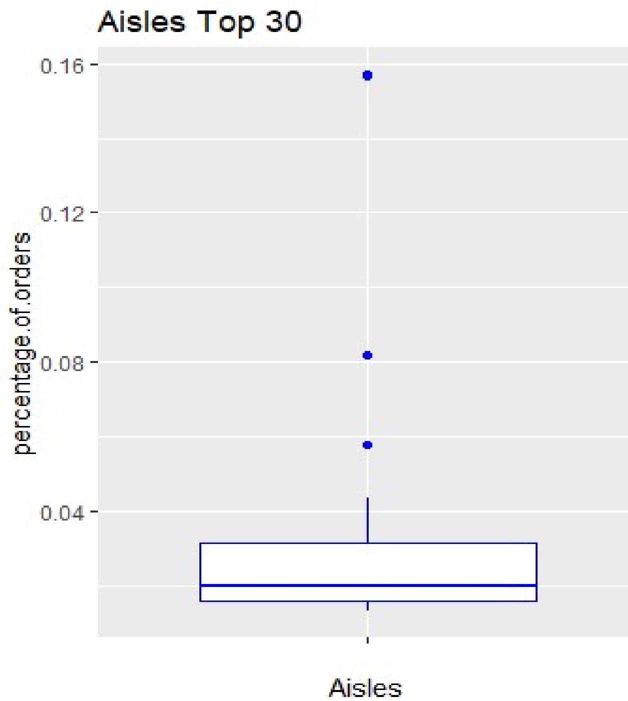

Certain departments are clearly more popular, like produce and dairy eggs. Both deparments combined contributed to more than 40% of total orders.

## Most Popular Aisles Sold

```
> tmp = order_products %>%
+   left_join(products) %>%
+   left_join(aisle) %>%
+   group_by(aisle) %>%
+   summarize(count=n()) %>%
+   top_n(n=30, wt=count) %>%  mutate(percentage=count/sum(count))
Joining, by = "product_id"
Joining, by = "aisle_id"
> p1 = ggplot (tmp, aes(x=reorder(aisle,count), y=percentage)) +
+   geom_col(fill="purple") + ggtitle('Aisles Top 30') + ylab('Percentage of 
Orders') +
+   theme (
+     axis.text.x=element_text(angle=90, hjust=1, vjust=0.5),
+     axis.title.x = element_blank()) +  ylab('Percentage of Orders') + xlab(
'Aisles')
> p2 = ggplot (tmp, aes( x= '', y=percentage )) +
+   ggtitle('Aisles Top 30') + ylab('percentage.of.orders') + geom_boxplot(co
lor="blue") + xlab('Aisles')
> grid.arrange(p1, p2, ncol = 2)
```

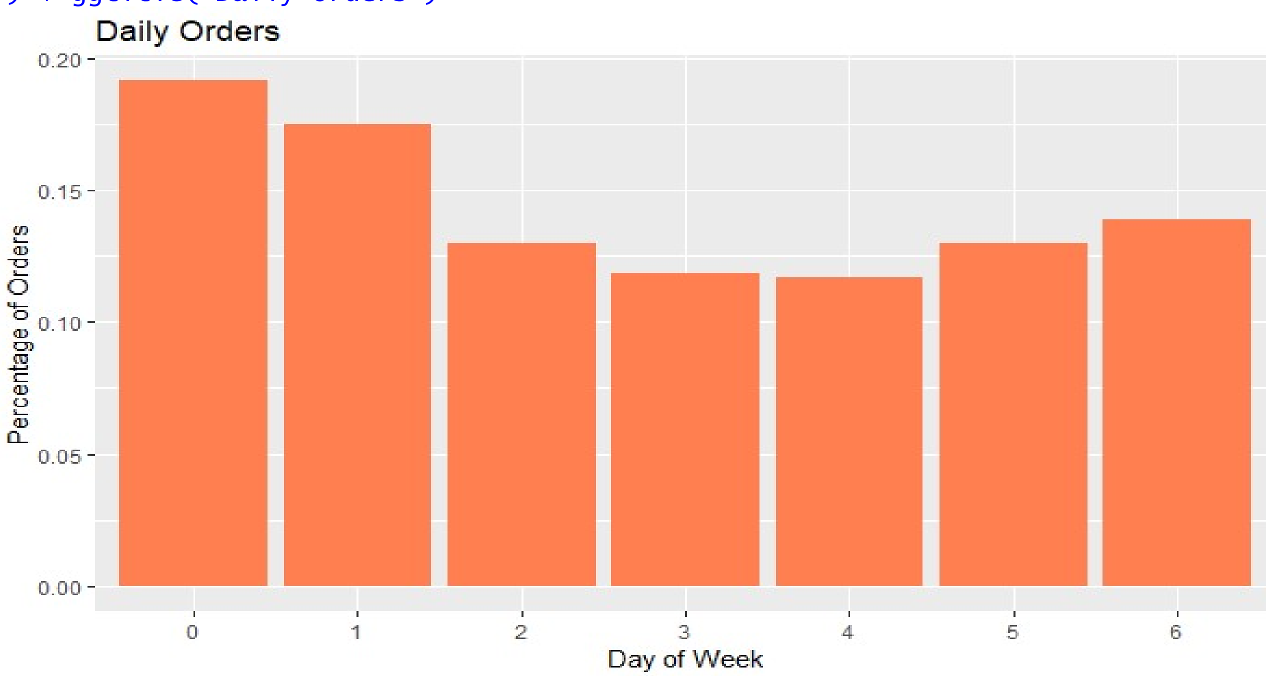
Looking at the products sold by aisle, we notice that certain aisle like vegetables and fruits contributes to almost 30% of total orders.




## Products ordered by day

```
> order_products_prior %>%
+   left_join(orders) %>%
+   group_by(order_dow) %>%
+   summarize(count = n()) %>%
+   mutate(percentage=count/sum(count)) %>%
+   ggplot (aes(x=as.factor(order_dow), y=percentage)) +
+     geom_col(fill="coral")+ xlab("Day of Week")+ ylab('Percentage of Orders
') + ggtitle('Daily Orders')
```



As we can see, both Day 0 and Day 1 stands out to be the busiest shopping day for instacart. This means that day of order made may influence the basket size.

```
> order_products_prior %>%
+   left_join(orders) %>% left_join(products) %>%
+   group_by(order_dow, product_name) %>%
+   summarize(n=n()) %>%
+   mutate(percentage=n/sum(n)) %>%
+   top_n(10, wt=n) %>%
+   ggplot (aes(x=as.factor(order_dow), y=percentage, fill=product_name)) +
+     geom_col() + ylab('Proprtion of Orders') + ggtitle('Daily Top 10 Produc
ts Ordered') +
+     theme(legend.position="bottom",legend.direction="horizontal")
```


Looking at the products ordered daily at Instacart, the top ten products ordered daily contributes between 7% to 8%.

Limes are part of top ten for Day 0 and Day 6, but not other days. Whereas Organic Whole Milk doesn't make it to top ten for Day 0. Organic. Raspberries does not make it to top 10 of Day 6. This means that there is a chance of predictability based on the day order is made.

# Data Analysis II

## Up and Running with Apriori Algorithm

As we mentioned, the “A priori” principle states that all subsets of a frequent itemset must also be frequent. In other words, if {A, B} is frequent, then {A} and {B} must both be frequent. Therefore, if we know that {A} does not meet a desired support threshold, there is no reason to consider {A, B} or any itemset containing {A}; it cannot possibly be frequent.

The Apriori algorithm uses this logic to exclude potential association rules prior to evaluating them. It creates association rules in the following two stage process:

1. Identifying all the item sets that meet a minimum support threshold.
2. Creating rules from these item sets using those meeting a minimum confidence threshold.

## Data pre-processing- Feature Engineering & creating a sparse matrix for transaction data

## Feature Engineering

Unlike the usual data frames where rows indicated example instances and columns indicated features, transactional data is somewhat different. For transactional data each row in the data specifies a single example—in this case, a transaction. However, rather than having a set number of features, each record comprises a comma-separated list of any number of items, from one to many. The features may differ from example to example.

We also use the left_join() function from Dplyr in R to join the columns in the different csv files. We use group_by function in R which collapses the unit of analysis from the complete dataset to individual groups. We  also use the summarize() function to collapse the Dataframe to a single row.

```
> basket_data = left_join(order_products_prior, products, by='product_id')
> head(basket_data)
  order_id product_id add_to_cart_order reordered          product_name
1        2      33120                 1         1    Organic Egg Whites
2        2      28985                 2         1 Michigan Organic Kale
3        2       9327                 3         0         Garlic Powder
4        2      45918                 4         1        Coconut Butter
5        2      30035                 5         0     Natural Sweetener
6        2      17794                 6         1               Carrots
  aisle_id department_id
1       86            16
2       83             4
```

```
3       104             13
4        19             13
5        17             13
6        83              4
> basket_data = group_by(basket_data, order_id)
> basket_data=summarise(basket_data,items=as.vector(list(product_name)))
> View(basket_data)
```

| | order_id | items |
|---|---|---|
| 1 | 2 | c("Organic Egg Whites", "Michigan Organic Kale", "Garlic Po... |
| 2 | 3 | c("Total 2% with Strawberry Lowfat Greek Strained Yogurt", "... |
| 3 | 4 | c("Plain Pre-Sliced Bagels", "Honey/Lemon Cough Drops", "... |
| 4 | 5 | c("Bag of Organic Bananas", "Just Crisp, Parmesan", "Fresh F... |
| 5 | 6 | c("Cleanse", "Dryer Sheets Geranium Scent", "Clean Day Lav... |
| 6 | 7 | c("Orange Juice", "Pineapple Chunks") |
| 7 | 8 | Original Hawaiian Sweet Rolls |
| 8 | 9 | c("Organic Red Radish, Bunch", "Whole White Mushrooms", ... |
| 9 | 10 | c("Banana", "Baby Portabella Mushrooms", "Organic Cilantro... |
| 10 | 11 | c("Teriyaki & Pineapple Chicken Meatballs", "Mango Pineap... |

## Creating a sparse matrix for transaction data

In order to create the sparse matrix data structure from the transactional data, we can use the functionality provided by the arules package. We Install and load the package as follows:

```
> install.packages("arules")
> library(arules)
> transactions=as(basket_data$items, 'transactions')
> head(transactions)
transactions in sparse format with
 6 transactions (rows) and
 49677 items (columns)
```

To see some basic information about the “cart” matrix we just created, we use the summary() function on the object:

```
> summary(transactions)
transactions as itemMatrix in sparse format with
 3214874 rows (elements/itemsets/transactions) and
 49677 columns (items) and a density of 0.0002030896

most frequent items:
                Banana Bag of Organic Bananas   Organic Strawberries
                472565                 379450                 264683
  Organic Baby Spinach    Organic Hass Avocado                (Other)
```

```
               241921                      213584                    30862286

element (itemset/transaction) length distribution:
sizes
     1       2       3       4       5       6       7       8       9      10      11 
12      13 
156748  186993  207027  222081  228330  227675  220006  203374  184347  165550  147461 
131580  116871 
    14      15      16      17      18      19      20      21      22      23      24 
25      26 
103683   91644   81192   71360   62629   54817   48096   41863   36368   31672   27065 
23613   20283 
    27      28      29      30      31      32      33      34      35      36      37 
38      39 
 17488   15102   13033   11251    9571    8035    6991    6041    5164    4407    3681 
3169    2653 
    40      41      42      43      44      45      46      47      48      49      50 
51      52 
  2272    1978    1642    1412    1227    1048     895     743     608     563     491 
394     348 
    53      54      55      56      57      58      59      60      61      62      63 
64      65 
   288     275     224     175     159     165     119     121     100      79      67 
57      53 
    66      67      68      69      70      71      72      73      74      75      76 
77      78 
    49      44      39      24      33      30      23      22      24       9      11 
15       9 
    79      80      81      82      83      84      85      86      87      88      89 
90      91 
     7       7       5       9       4      10       5       7       4       7       4 
1       4 
    92      93      94      95      96      98      99     100     101     102     104 
105     108 
     9       4       1       4       3       4       2       4       2       3       2 
1       2 
   109     112     114     115     116     121     127     137     145 
     2       1       1       1       1       1       1       1       1 

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    5.00    8.00   10.09   14.00  145.00 

includes extended item information - examples:

includes extended item information - examples:
```

The output 3,214,874 rows refer to the number of transactions, and the output 49,677 columns refer to the 49,677 different items that might appear in someone's grocery basket.

The density value refers to the proportion of nonzero matrix cells. As seen there are very few non-zero matrix cells. Most cells are zeros hence the name “sparse matrix”.

The next block of the summary () output lists the items that were most commonly found in the transactional data.

Bananas were the most frequently purchased item accounting for ((472565/ 3,214,874) *100) 14.7 percent of the transactions. The other frequent items were Organic Bananas, Organic Strawberries, organic Baby Spinach and Large Lemons.

Next, we see the size of the transactions. A total of 156,748 transactions contained only a single item, while one transaction had 145 items.

The first quartile and median purchase sizes are 5 items and 8 items. The mean number per transaction was 10 items.

The maximum number of items bought per transaction was 145.

## Generating Rules with the Apriori algorithm

The general syntax of the apriori algorithm is as follows:

apriori (data, list (support=, confidence=, minlen=))

where data is the sparse matrix holding the transactional data. Support specifies the minimum required rule support and confidence specifies the minimum required rule confidence. Minlen specifies the minimum number of rule items

There can sometimes be a fair amount of trial and error needed to find the support and confidence parameters that produce a reasonable number of association rules. Setting the minimum support parameters too high might find no rules or rules that are too generic to be very useful. On the other hand, a threshold too low might result in an unwieldy number of rules or cause a slowdown of the system.

Setting the minimum confidence involves a delicate balance too. If confidence is too low, we might be overwhelmed with a large number of unreliable rules. If we set confidence too high, we will be limited to the rules that are obvious or inevitable.

We'll start with a confidence threshold of 0.30, which means that in order to be included in the results, the rule has to be correct at least 30 percent of the time. We also set minlen = 2 to eliminate rules that contain fewer than two items.

```
> basket_rules <- apriori(transactions, parameter = list(support = 0.0005,
confidence = 0.30, minlen = 2))
> basket_rules
set of 739 rules
```

To obtain a closer view of the association rules, we can use the summary () function as follows.

```
> summary(basket_rules)
```

```
set of 739 rules

rule length distribution (lhs + rhs):sizes
  2   3   4 
125 567  47 

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  2.000   3.000   3.000   2.894   3.000   4.000 

summary of quality measures:
    support            confidence          lift              count      
 Min.   :0.0005008   Min.   :0.3001   Min.   :  2.042   Min.   : 1610  
 1st Qu.:0.0005733   1st Qu.:0.3242   1st Qu.:  2.626   1st Qu.: 1843  
 Median :0.0007030   Median :0.3524   Median :  3.110   Median : 2260  
 Mean   :0.0009536   Mean   :0.3699   Mean   : 21.830   Mean   : 3066  
 3rd Qu.:0.0009458   3rd Qu.:0.3991   3rd Qu.:  5.213   3rd Qu.: 3040  
 Max.   :0.0166087   Max.   :0.6482   Max.   :327.306   Max.   :53395  

mining info:
```

The rule length distribution tells us how many rules have each count of items. In our rule set, 125 rules have only two items, while 567 have three, and 47 have four.

Next, we see the summary statistics of the rule quality measures: support, confidence, lift and count.

The lift of a rule measures how much more likely one item or itemset is purchased relative to its typical rate of purchase, given that you know another item or itemset has been purchased. This is defined by the following equation:

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(X)}$$

A large lift value is therefore a strong indicator that a rule is important, and reflects a true connection between the items.

We can take a look at specific rules using the inspect () function. For instance, we can look at the first 5 rules as follows:

```
> inspect(basket_rules[1:5])
    lhs                                                    rhs                                        
support confidence       lift count
[1] {Organic Fuji Apples}                              => {Bag of Organic Bananas
}                            0.0005468333  0.3181325   2.695364  1758
[2] {Zero Calorie Cola}                                => {Soda}
0.0012348851  0.4638934  41.668546  3970
[3] {YoKids Squeeze Organic Blueberry Blue Yogurt} => {YoKids Squeeze! Organi
c Strawberry Flavor Yogurt} 0.0005269258  0.4870615 230.034001  1694
[4] {Red Delicious Apple}                              => {Banana}
0.0005502548  0.4179069   2.843033  1769
[5] {Blueberries Pint}                                 => {Banana}
0.0005259926  0.3733716   2.540058  1691
```

We can read the first rule as follows: "if a customer buys Organic Fuji Apples, they will also buy a bag of Organic Bananas." With support of 0.0005 and confidence of 0.318, we can determine that this rule covers 0.05 percent of the transactions and is correct in 31.8 percent of purchases involving Organic Fuji Apples.

The lift value tells us how much more likely a customer is to buy a bag of Organic Bananas relative to the average customer, given that he or she bought.

## Sorting the set of association rules

Occasionally, the most useful rules might be the ones with the highest support, confidence, or lift. The arules package includes a sort() function that can be used to reorder the list of rules so that the ones with the highest or lowest values of the quality measure come first. To reorder, we can apply sort () while specifying a "support", "confidence", or "lift" value to the by parameter.

For instance, the best five rules according to the lift statistic can be examined using the following command:

```
> inspect(sort(basket_rules, by='lift')[1:5])
    lhs                                rhs                                s
upport confidence     lift count
[1] {Almond Milk Blueberry Yogurt}  => {Almond Milk Peach Yogurt}      0.0007
029824  0.4792197 327.3063  2260
[2] {Almond Milk Peach Yogurt}      => {Almond Milk Blueberry Yogurt}  0.0007
029824  0.4801360 327.3063  2260
[3] {Almond Milk Strawberry Yogurt} => {Almond Milk Blueberry Yogurt}  0.0008
404684  0.4724602 322.0738  2702
[4] {Almond Milk Blueberry Yogurt}  => {Almond Milk Strawberry Yogurt} 0.0008
404684  0.5729432 322.0738  2702
[5] {Almond Milk Strawberry Yogurt} => {Almond Milk Peach Yogurt}      0.0007
704812  0.4331177 295.8187  2477
```

The first rule, with a lift of about 327.31, implies that people who buy Almond Milk Blueberry Yogurt are nearly three hundred and twenty seven times more likely to buy Almond Milk Peach Yogurt than the typical customer.

Perhaps, the store manager should consider placing these two yogurt flavours next to each other.

## Taking subsets of association rules

We may need to find whether the Almond Milk Blueberry Yogurt is purchased with other items.

We'll need to find all the rules that include Almond Milk Blueberry Yogurt in some form. The subset () function provides a method to search for subsets of transactions, items, or rules.

To use it to find any rules with Almond Milk Blueberry Yogurt appearing in the rule, use the following command.

```
> berryrules <- subset(basket_rules, items %in% "Almond Milk Blueberry Yogurt
")
> inspect(berryrules)
    lhs                                   rhs                                  s
upport confidence     lift count
[1] {Almond Milk Blueberry Yogurt}  => {Almond Milk Peach Yogurt}      0.0007
029824  0.4792197 327.3063  2260
[2] {Almond Milk Peach Yogurt}      => {Almond Milk Blueberry Yogurt}  0.0007
029824  0.4801360 327.3063  2260
[3] {Almond Milk Blueberry Yogurt}  => {Almond Milk Strawberry Yogurt} 0.0008
404684  0.5729432 322.0738  2702
[4] {Almond Milk Strawberry Yogurt} => {Almond Milk Blueberry Yogurt}  0.0008
404684  0.4724602 322.0738  2702
```

If we are interested in those rules whose confidence level is above 0.6, we could subset them as follows:

```
> rules<- subset(basket_rules,confidence > 0.6)
> inspect(rules)
    lhs                                                         rhs
support confidence      lift count
[1] {Non Fat Acai & Mixed Berries Yogurt,
     Non Fat Raspberry Yogurt}                              => {Icelandic Sty
le Skyr Blueberry Non-fat Yogurt}         0.0005878924  0.6009539 100.32669
1890
[2] {Sparkling Lemon Water,
     Sparkling Water Berry}                                 => {Sparkling Wat
er Grapefruit}                            0.0007113809  0.6059883  25.67241
2287
[3] {Lime Sparkling Water,
     Sparkling Water Berry}                                 => {Sparkling Wat
er Grapefruit}                            0.0008267820  0.6321046  26.77881
2658
[4] {Peach Pear Flavored Sparkling Water,
     Sparkling Lemon Water}                                 => {Sparkling Wat
er Grapefruit}                            0.0006485480  0.6362527  26.95454
2085
[5] {Lime Sparkling Water,
     Peach Pear Flavored Sparkling Water}                   => {Sparkling Wat
er Grapefruit}                            0.0008504221  0.6481745  27.45960
2734
[6] {Total 2% All Natural Greek Strained Yogurt with Honey,
     Total 2% Lowfat Greek Strained Yogurt with Peach}      => {Total 2% with
Strawberry Lowfat Greek Strained Yogurt} 0.0006998719  0.6053269  65.10051  2
250
[7] {Total 2% All Natural Greek Strained Yogurt with Honey,
     Total 2% Lowfat Greek Strained Yogurt With Blueberry}  => {Total 2% with
Strawberry Lowfat Greek Strained Yogurt} 0.0007480853  0.6141471  66.04909  2
405
```