

# ML1000 Assignment 3

Anupama r.k, Queenie Tsang, Crystal (Yunan) Zhu

09/03/2021

## Abstract

The main goal of recommender systems is to provide suggestions to online users to make better decisions from many alternatives available over the Web. A better recommender system is directed more towards personalized recommendations by taking into consideration information about a product, such as specifications, users' purchase history, comparison with other products, and so on, before making recommendations.

## Business Case

## Data Understanding

Instacart Market Basket Analysis dataset obtained from <https://www.kaggle.com/c/instacart-market-basket-analysis/data> (<https://www.kaggle.com/c/instacart-market-basket-analysis/data>)

### How do we merge the data files?

There are six data files, excluding the sample\_submission.csv file, from the Instacart Market Basket Analysis data - aisles.csv, departments.csv, order\_products\_\_train.csv, order\_products\_\_prior.csv, orders.csv and products.csv.

### Data file descriptions

aisles.csv - contains aisle id and aisle description columns

departments.csv - contains department id and department description columns

order\_products\_\_\*.csv - These files specify which products were purchased in each order. order\_products\_\_prior.csv contains previous order contents for all customers. 'reordered' indicates that the customer has a previous order that contains the product. order\_products\_\_train.csv contains order information for transactions which will be used for training the model.

orders.csv - This file tells to which set (prior, train, test) an order belongs. You are predicting reordered items only for the test set orders. 'order\_dow' is the day of week.

products.csv - contains product\_id, product\_name, aisle\_id, department\_id

### Steps to merge the data files:

1. Merged the aisles data with the products data to obtain Merged dataset 1, so that we know which aisle each product belongs to.

2. Combined the Merged dataset 1 with the department data to obtain Merged dataset 2, so we know which aisle and department each product is from.
3. Add Merged dataset 2, which contains product full information, to order\_products\_\_train and order\_products\_\_prior files, respectively, to obtain Merged dataset 3 (Train) and Merged dataset 4 (Prior), so that we know the product information (e.g. product names, aisles and departments they belong to) of the products in the training and prior orders.

Read in the merged training data set:

```
X <- read.csv("C:/Users/qt09n/Desktop/Project/orders_TRAIN_products_MERGED.csv")
```

```
str(X)
```

```
## 'data.frame': 1384617 obs. of 16 variables:
## $ order_id : num 1 1 1 1 1 1 1 1 36 36 ...
## $ user_id : num 112108 112108 112108 112108 112108 ...
## $ eval_set : chr "train" "train" "train" "train" ...
## $ order_number : int 4 4 4 4 4 4 4 4 23 23 ...
## $ order_dow : int 4 4 4 4 4 4 4 4 6 6 ...
## $ order_hour_of_day : int 10 10 10 10 10 10 10 10 18 18 ...
## $ days_since_prior_order: int 9 9 9 9 9 9 9 9 30 30 ...
## $ product_id : int 10246 11109 22035 47209 13176 49302 49683 43633 39612 46620 ...
## $ add_to_cart_order : int 3 2 8 7 6 1 4 5 1 5 ...
## $ reordered : int 0 1 1 0 0 1 0 1 0 1 ...
## $ product_name : chr "Organic Celery Hearts" "Organic 4% Milk Fat Whole Milk Cottage Chee
## $ aisle : chr "fresh vegetables" "other creams cheeses" "packaged cheese" "fresh f
## $ department : chr "produce" "dairy eggs" "dairy eggs" "produce" ...
## $ aisle_id : int 83 108 21 24 24 120 83 95 2 86 ...
## $ department_id : int 4 16 16 4 4 16 4 15 16 16 ...
## $ X_merge : int 3 3 3 3 3 3 3 3 3 3 ...
```

```
dim(X)
```

```
## [1] 1384617 16
```

Check missing values: complete.cases will return a logical vector indicating which rows have no missing values. Then use the vector to get only complete rows with X[,]

```
X <- X[complete.cases(X),]
```

```
sum(is.na(X))
```

```
## [1] 0
```

No missing values in dataset.

look at first 10 rows of dataset

```
head(X)
```

```
## order_id user_id eval_set order_number order_dow order_hour_of_day
## 1 1 112108 train 4 4 10
## 2 1 112108 train 4 4 10
## 3 1 112108 train 4 4 10
## 4 1 112108 train 4 4 10
## 5 1 112108 train 4 4 10
## 6 1 112108 train 4 4 10
## days_since_prior_order product_id add_to_cart_order reordered
## 1 9 10246 3 0
## 2 9 11109 2 1
## 3 9 22035 8 1
## 4 9 47209 7 0
## 5 9 13176 6 0
## 6 9 49302 1 1
## product_name aisle department
## 1 Organic Celery Hearts fresh vegetables produce
## 2 Organic 4% Milk Fat Whole Milk Cottage Cheese other creams cheeses dairy eggs
## 3 Organic Whole String Cheese packaged cheese dairy eggs
## 4 Organic Hass Avocado fresh fruits produce
## 5 Bag of Organic Bananas fresh fruits produce
## 6 Bulgarian Yogurt yogurt dairy eggs
## aisle_id department_id X_merge
## 1 83 4 3
## 2 108 16 3
## 3 21 16 3
## 4 24 4 3
## 5 24 4 3
## 6 120 16 3
```

```
summary(X)
```

```
## order_id user_id eval_set order_number
## Min. : 1 Min. : 1 Length:1384617 Min. : 4.00
## 1st Qu.: 843370 1st Qu.: 51732 Class :character 1st Qu.: 6.00
## Median :1701880 Median :102933 Mode :character Median : 11.00
## Mean :1706298 Mean :103113 Mean : 17.09
## 3rd Qu.:2568023 3rd Qu.:154959 3rd Qu.: 21.00
## Max. :3421070 Max. :206209 Max. :100.00
## order_dow order_hour_of_day days_since_prior_order product_id
## Min. :0.000 Min. : 0.00 Min. : 0.00 Min. : 1
## 1st Qu.:1.000 1st Qu.:10.00 1st Qu.: 7.00 1st Qu.:13380
## Median :3.000 Median :14.00 Median :15.00 Median :25298
## Mean :2.701 Mean :13.58 Mean :17.07 Mean :25556
## 3rd Qu.:5.000 3rd Qu.:17.00 3rd Qu.:30.00 3rd Qu.:37940
## Max. :6.000 Max. :23.00 Max. :30.00 Max. :49688
## add_to_cart_order reordered product_name aisle
## Min. : 1.000 Min. :0.0000 Length:1384617 Length:1384617
## 1st Qu.: 3.000 1st Qu.:0.0000 Class :character Class :character
## Median : 7.000 Median :1.0000 Mode :character Mode :character
## Mean : 8.758 Mean :0.5986
## 3rd Qu.:12.000 3rd Qu.:1.0000
## Max. :80.000 Max. :1.0000
## department aisle_id department_id X_merge
## Length:1384617 Min. : 1.0 Min. : 1.00 Min. :3
```

```
## Class :character 1st Qu.: 31.0 1st Qu.: 4.00 1st Qu.:3
## Mode :character Median : 83.0 Median : 8.00 Median :3
## Mean : 71.3 Mean : 9.84 Mean :3
## 3rd Qu.:107.0 3rd Qu.:16.00 3rd Qu.:3
## Max. :134.0 Max. :21.00 Max. :3
```

Check total users:

```
user_count <- unique(X$user_id)
length(user_count)
```

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

Change the character columns to factors and create a new column using mutate:

```
X$product_name <- as.factor(X$product_name)
X$department<- as.factor(X$department)
X$aisle <- as.factor(X$aisle)
```

```
unique(X$aisle)
```

```
## [1] fresh vegetables      other creams cheeses
## [3] packaged cheese       fresh fruits
## [5] yogurt                canned meat seafood
## [7] specialty cheeses     eggs
## [9] lunch meat            cream
## [11] water seltzer sparkling water packaged vegetables fruits
## [13] oils vinegars         fresh herbs
## [15] frozen produce        nuts seeds dried fruit
## [17] canned meals beans    food storage
## [19] baking ingredients    hot dogs bacon sausage
## [21] refrigerated          plates bowls cups flatware
## [23] butter                canned jarred vegetables
## [25] paper goods           fresh dips tapenades
## [27] soup broth bouillon   dish detergents
## [29] tortillas flat bread  condiments
## [31] milk                  soap
## [33] frozen meat seafood   soy lactosefree
## [35] canned fruit applesauce refrigerated pudding desserts
## [37] laundry              frozen appetizers sides
## [39] crackers             ice cream ice
## [41] juice nectars         chips pretzels
## [43] cold flu allergy      muscles joints pain relief
## [45] pasta sauce           bread
## [47] grains rice dried goods spreads
## [49] popcorn jerky         baby accessories
## [51] other                 missing
## [53] digestion            more household
## [55] packaged produce     breakfast bars pastries
## [57] candy chocolate       spices seasonings
## [59] cleaning products    diapers wipes
## [61] fresh pasta           frozen breakfast
```

```
## [63] asian foods           preserved dips spreads
## [65] latino foods          pickled goods olives
## [67] instant foods         energy granola bars
## [69] packaged meat         hot cereal pancake mixes
## [71] soft drinks          cookies cakes
## [73] frozen pizza          tea
## [75] prepared meals        energy sports drinks
## [77] poultry counter       trail mix snack mix
## [79] doughs gelatins bake mixes prepared soups salads
## [81] buns rolls           dry pasta
## [83] deodorants           cereal
## [85] frozen meals         breakfast bakery
## [87] white wines          coffee
## [89] fruit vegetable snacks oral hygiene
## [91] packaged seafood     bulk grains rice dried goods
## [93] packaged poultry     body lotions soap
## [95] tofu meat alternatives dog food care
## [97] bakery desserts      baby food formula
## [99] honeys syrups nectars meat counter
## [101] trash bags liners    kitchen supplies
## [103] hair care            beers coolers
## [105] first aid            vitamins supplements
## [107] granola              protein meal replacements
## [109] shave needs          salad dressing toppings
## [111] indian foods         frozen vegan vegetarian
## [113] spirits              frozen dessert
## [115] mint gum             cat food care
## [117] facial care          specialty wines champagnes
## [119] skin care            frozen breads doughs
## [121] red wines            marinades meat preparation
## [123] feminine care        baking supplies decor
## [125] ice cream toppings   seafood counter
## [127] cocoa drink mixes     kosher foods
## [129] air fresheners candles beauty
## [131] bulk dried fruits vegetables eye ear care
## [133] baby bath body care   frozen juice
## 134 Levels: air fresheners candles asian foods ... yogurt
```

```
#134 aisles
```

```
unique(X$department)
```

```
## [1] produce      dairy eggs    canned goods  deli
## [5] beverages    pantry        frozen        snacks
## [9] household    meat seafood  bakery        personal care
## [13] dry goods pasta babies        other         missing
## [17] breakfast    international alcohol       bulk
## [21] pets
## 21 Levels: alcohol babies bakery beverages breakfast bulk ... snacks
```

```
levels(X$department)
```

```
## [1] "alcohol"      "babies"      "bakery"      "beverages"
```

```
## [5] "breakfast"      "bulk"      "canned goods"  "dairy eggs"
## [9] "deli"           "dry goods pasta" "frozen"        "household"
## [13] "international"  "meat seafood"  "missing"       "other"
## [17] "pantry"         "personal care" "pets"          "produce"
## [21] "snacks"
```

```
#21 departments
```

```
length(unique(X$product_name))
```

```
## [1] 39123
```

```
#39123 products
```

Total products from product id column:

```
product_count <- unique(X$product_id)
length(product_count)
```

```
## [1] 39123
```

```
class(X$order_hour_of_day)
```

```
## [1] "integer"
```

```
#[1] "integer"
```

How many unique orders are in the training dataset?

```
length(unique(X$order_id))
```

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

How many users are in the training dataset?

```
length(unique(X$user_id))
```

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

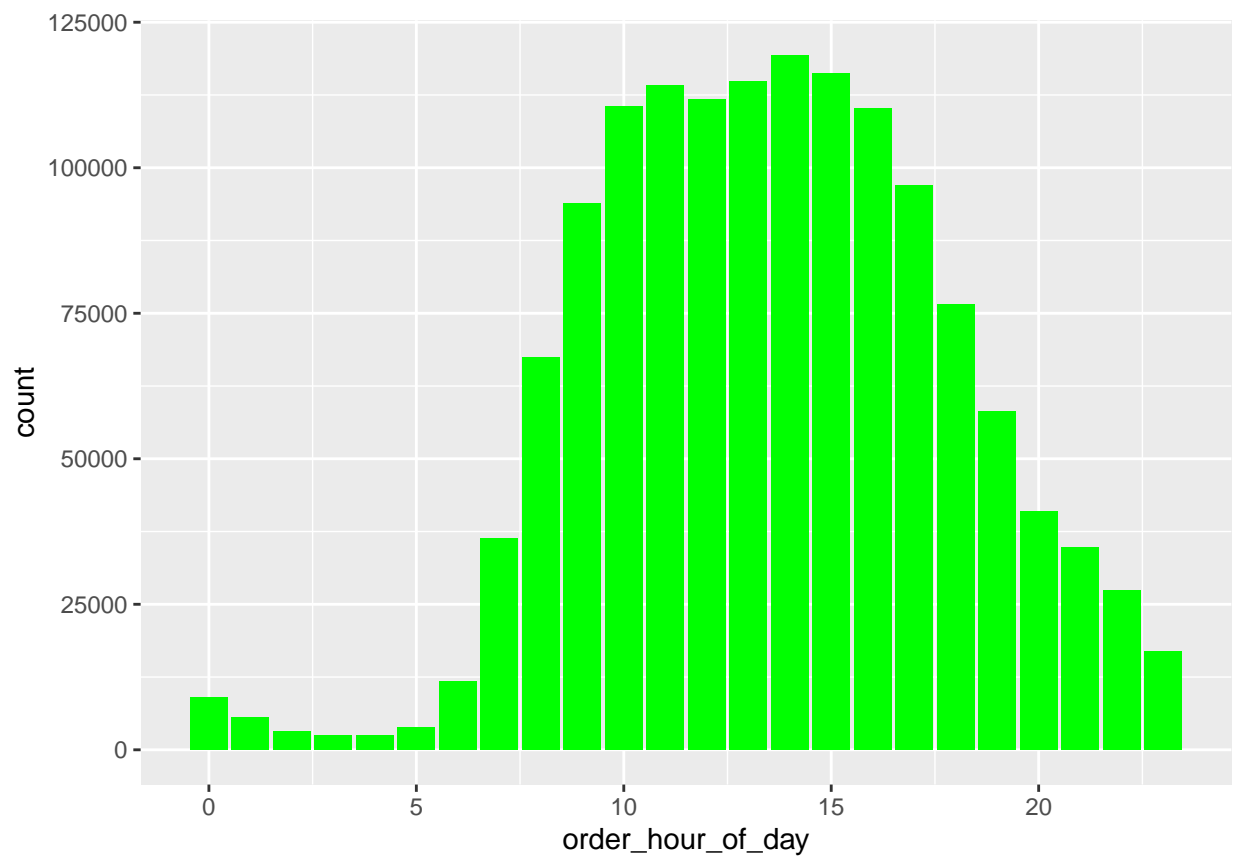
So it looks like the number of users are the same as the number of orders...

Recode order\_hour\_of\_day to numeric:

```
X$order_hour_of_day <- as.numeric(X$order_hour_of_day)
```

Look at when people order:

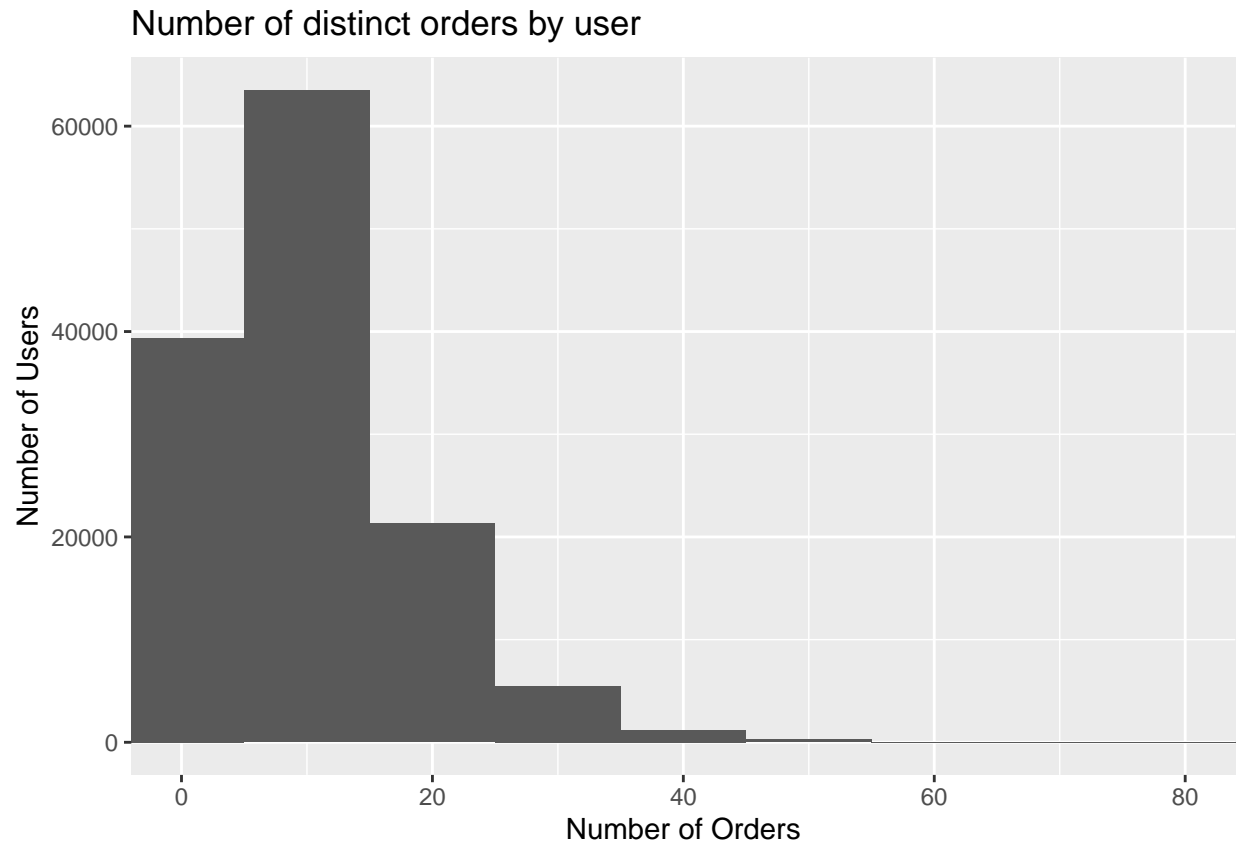
```
X %>% ggplot(aes(x= order_hour_of_day)) +  
  geom_bar(stat="count", fill="green")
```



People mostly order from 8:00 - 17:00 (8AM - 5PM).

Number of distinct orders by user:

```
p1<-X %>%  
  group_by(user_id) %>%  
  dplyr::summarise(count_order=n()) %>%  
  ungroup()  
  
ggplot(p1, aes(count_order)) + geom_histogram(binwidth = 10)+labs(title="Number of distinct orders by user",  
  x ="Number of Orders", y = "Number of Users")+coord_cartesian(xlim = c(0, 80))
```



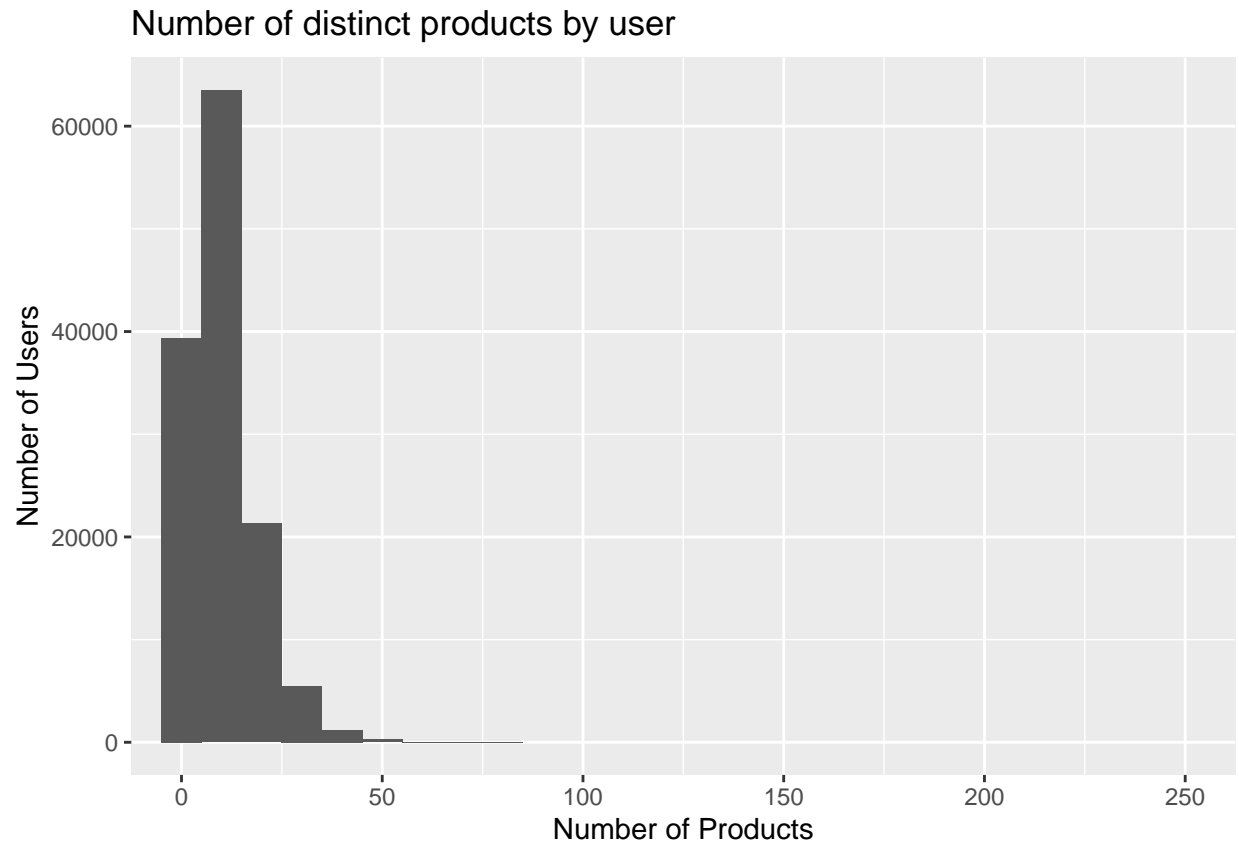
Number of distinct products per user:

```
p2 <- X %>%
  group_by(user_id, product_id) %>%
  dplyr::summarise(count3=n()) %>%
  select(user_id, product_id, count3) %>%
  ungroup() %>%
  group_by(user_id) %>%
  dplyr::summarise(count_product=n()) %>%
  ungroup()
```

## 'summarise()' has grouped output by 'user\_id'. You can override using the '.groups' argument.

```
ggplot(p2, aes(count_product)) + geom_histogram(binwidth = 10) + labs(title="Number of distinct products l
  x = "Number of Products", y = "Number of Users") + coord_cartesian(xlim = c(0, 250))
```



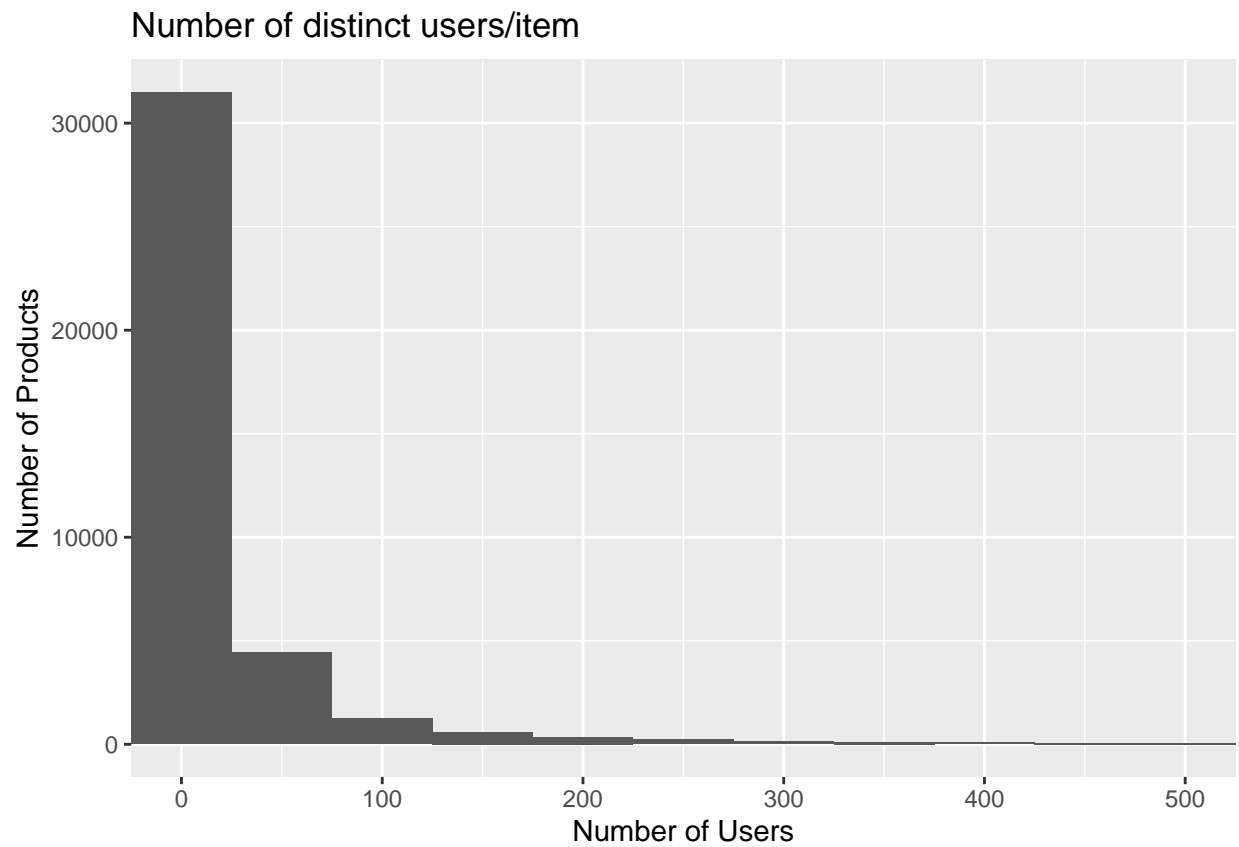


Number of distinct users/item

```
p3 <- X %>%
  group_by(product_id, user_id) %>%
  dplyr::summarise(count4=n()) %>%
  select(user_id, product_id, count4) %>%
  ungroup() %>%
  group_by(product_id) %>%
  dplyr::summarise(count_user=n()) %>%
  ungroup()
```

## 'summarise()' has grouped output by 'product\_id'. You can override using the '.groups' argument.

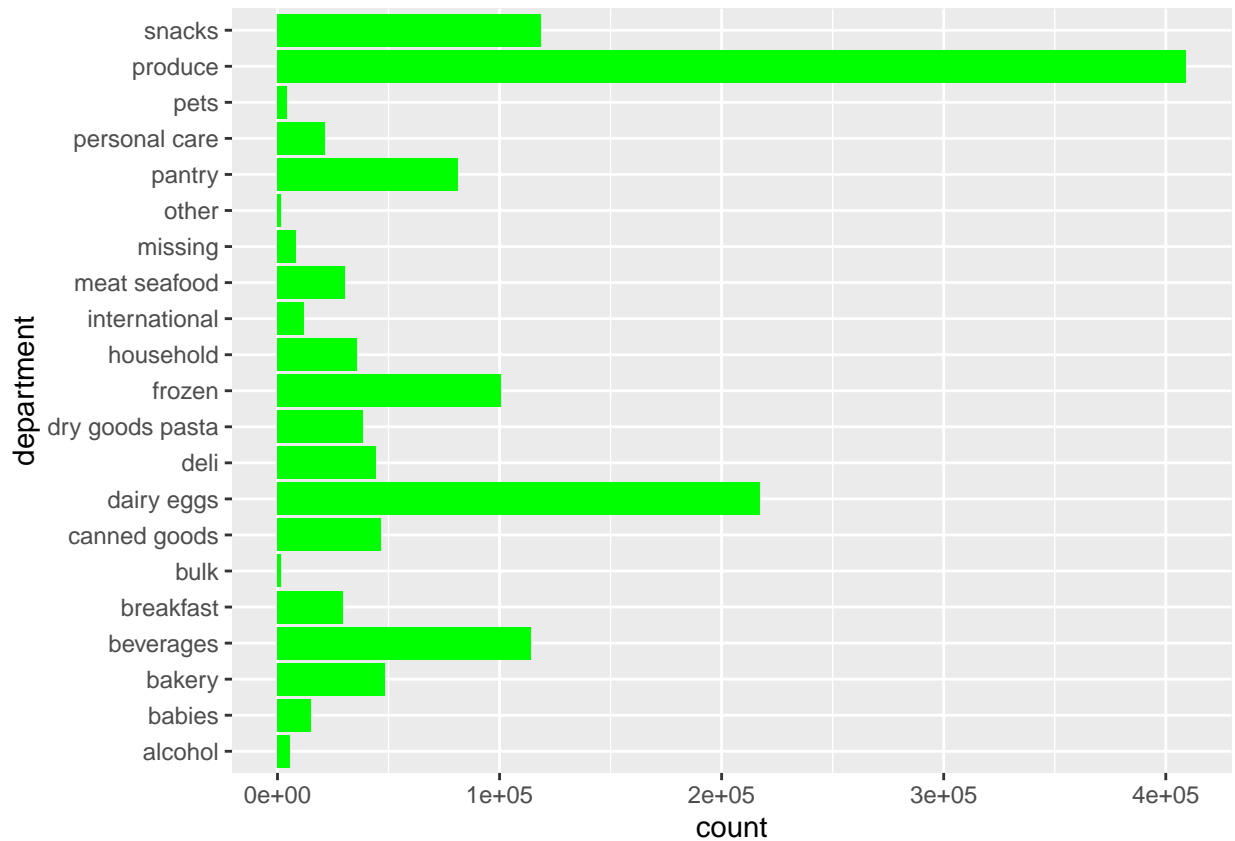
```
ggplot(p3, aes(count_user)) + geom_histogram(binwidth = 50) + labs(title="Number of distinct users/item",
  x = "Number of Users", y = "Number of Products") + coord_cartesian(xlim = c(0, 500))
```



Most frequently bought products

```
X %>% ggplot(aes(x= department)) +  
  geom_histogram(stat="count", fill="green")+  
  coord_flip()
```

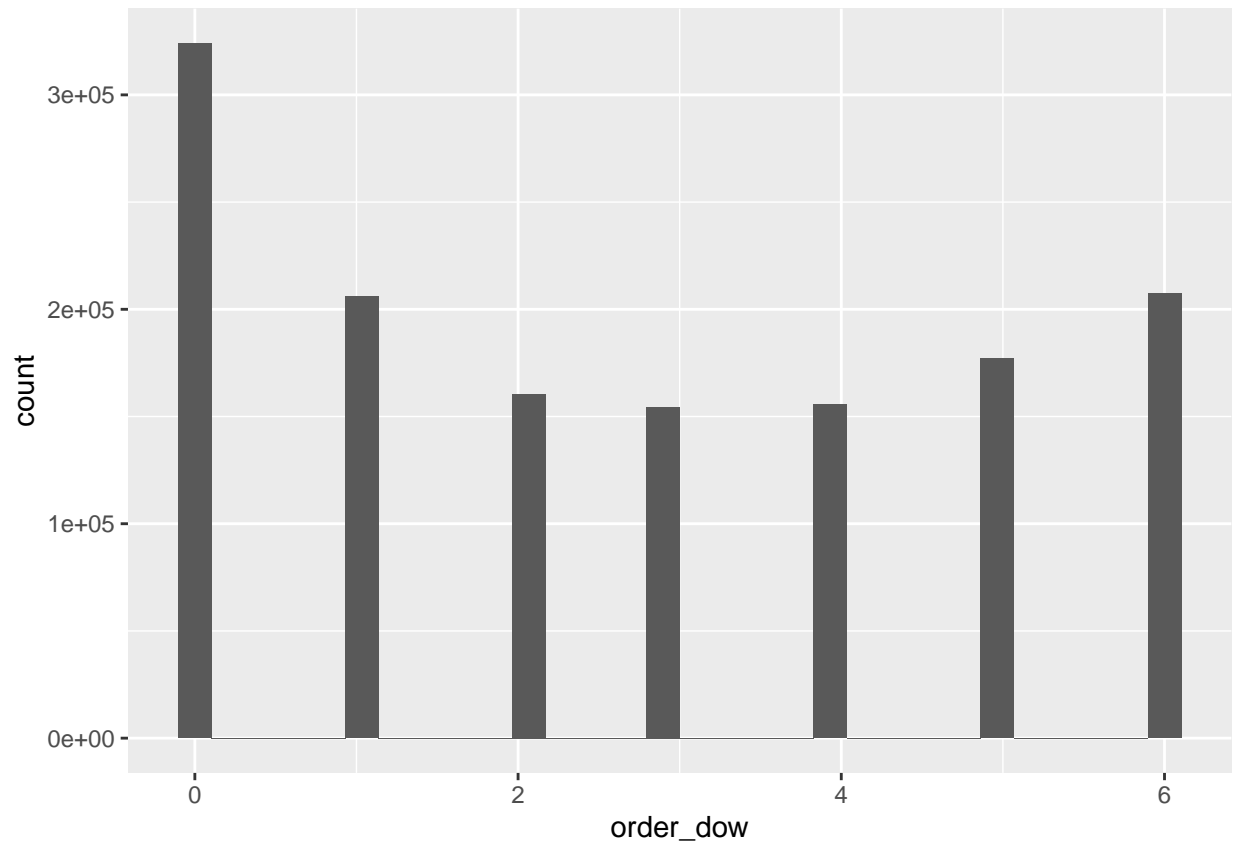
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



Most orders come from the produce aisle, with snacks and dairy, eggs among the top 3 department aisles. 'order\_dow' is the day of week. Which days are orders more commonly placed on?

```
X %>% ggplot(aes(x=order_dow))+
  geom_histogram()
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

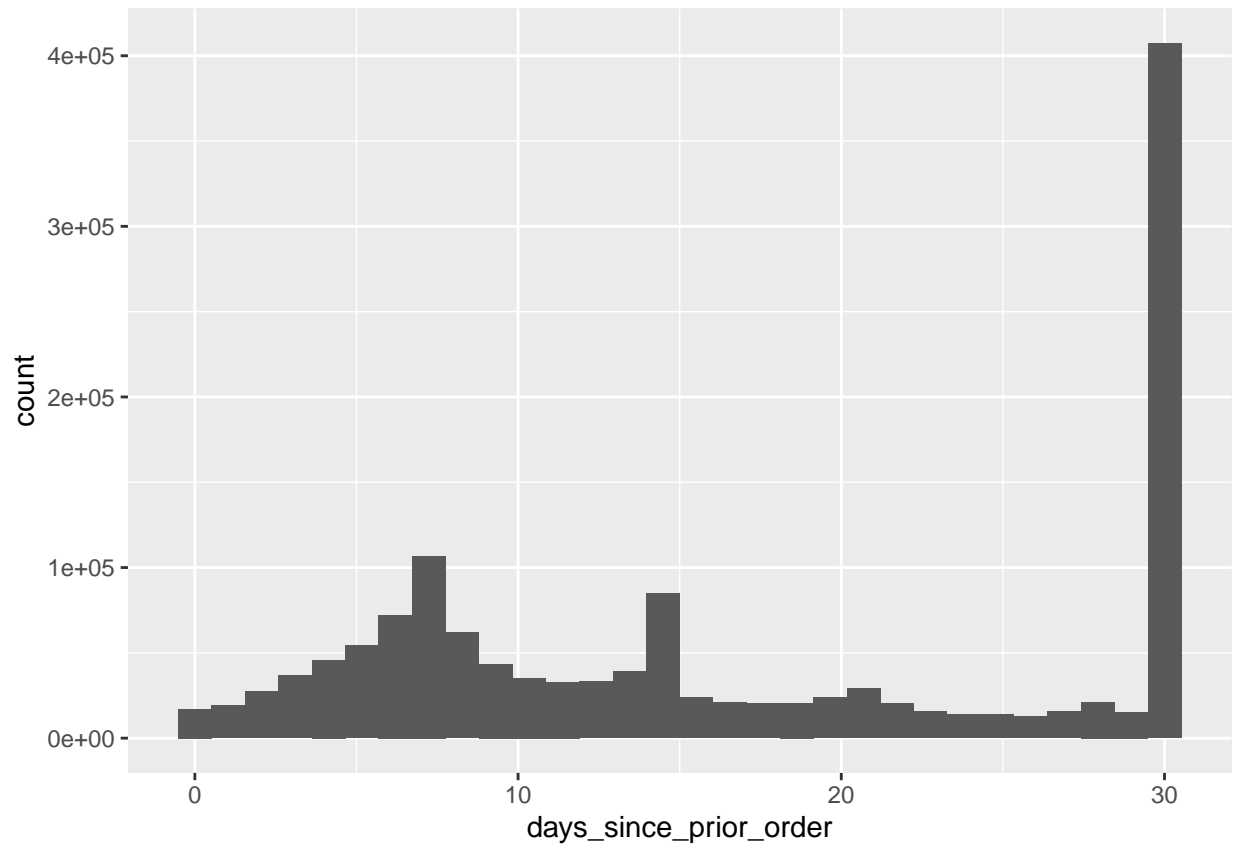


*#Sunday, Monday and Saturday appear to be the most common days where people place their orders.*

How many days pass between an order and the next order?

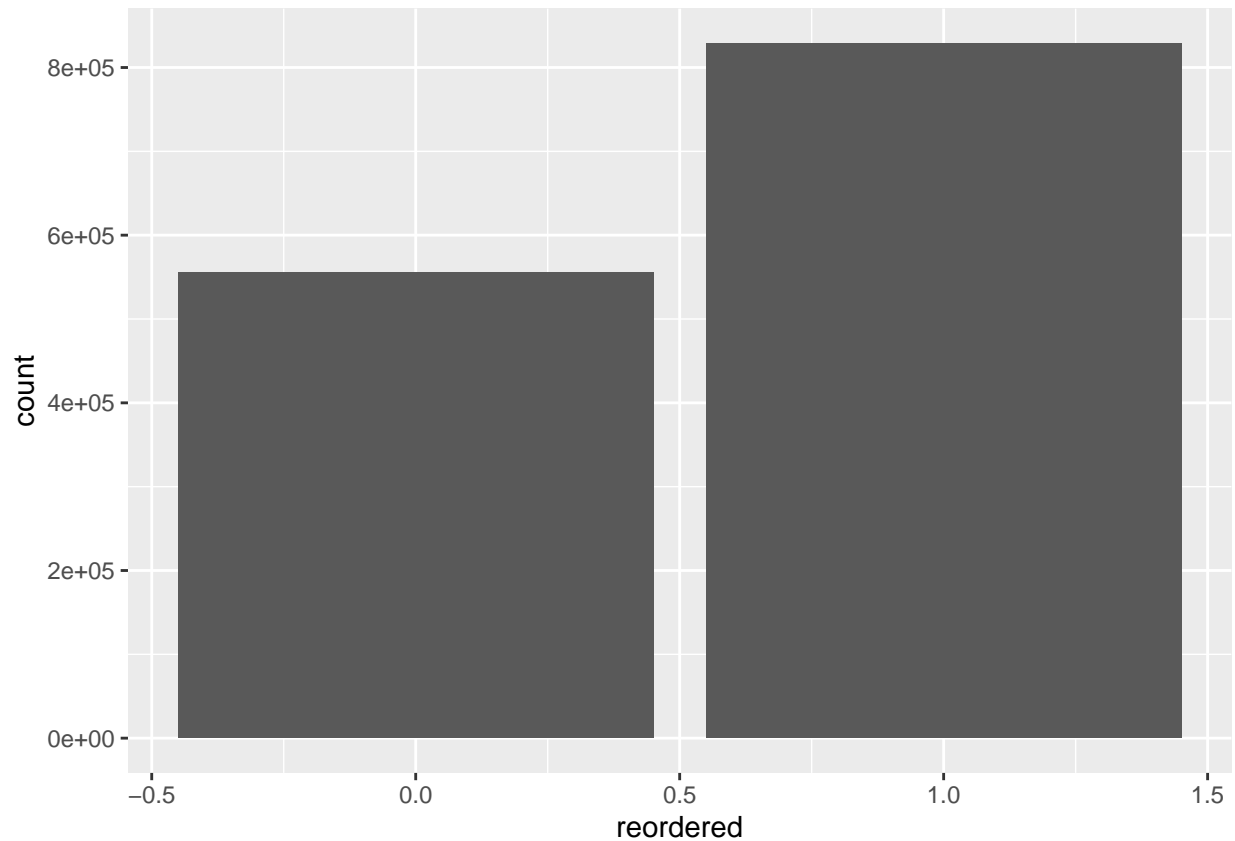
```
X %>% ggplot(aes(x=days_since_prior_order))+  
  geom_histogram()
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



People most commonly order again after 30 days (1 month), or 7 days (1 week).

```
ggplot(X) +  
geom_bar(mapping= aes(x=reordered))
```



Find the percentage of transactions from the top 10 products sold.

```
X %>%
  group_by(product_name) %>%
  dplyr::summarize(count = n()) %>%
  mutate(pct=(count/sum(count))*100) %>%
  arrange(desc(pct)) %>%
  ungroup() %>%
  top_n(10, wt=pct)
```

```
## # A tibble: 10 x 3
##   product_name      count  pct
##   <fct>          <int> <dbl>
## 1 Banana          18726  1.35
## 2 Bag of Organic Bananas 15480  1.12
## 3 Organic Strawberries 10894  0.787
## 4 Organic Baby Spinach  9784  0.707
## 5 Large Lemon       8135  0.588
## 6 Organic Avocado    7409  0.535
## 7 Organic Hass Avocado 7293  0.527
## 8 Strawberries      6494  0.469
## 9 Limes             6033  0.436
## 10 Organic Raspberries 5546  0.401
```

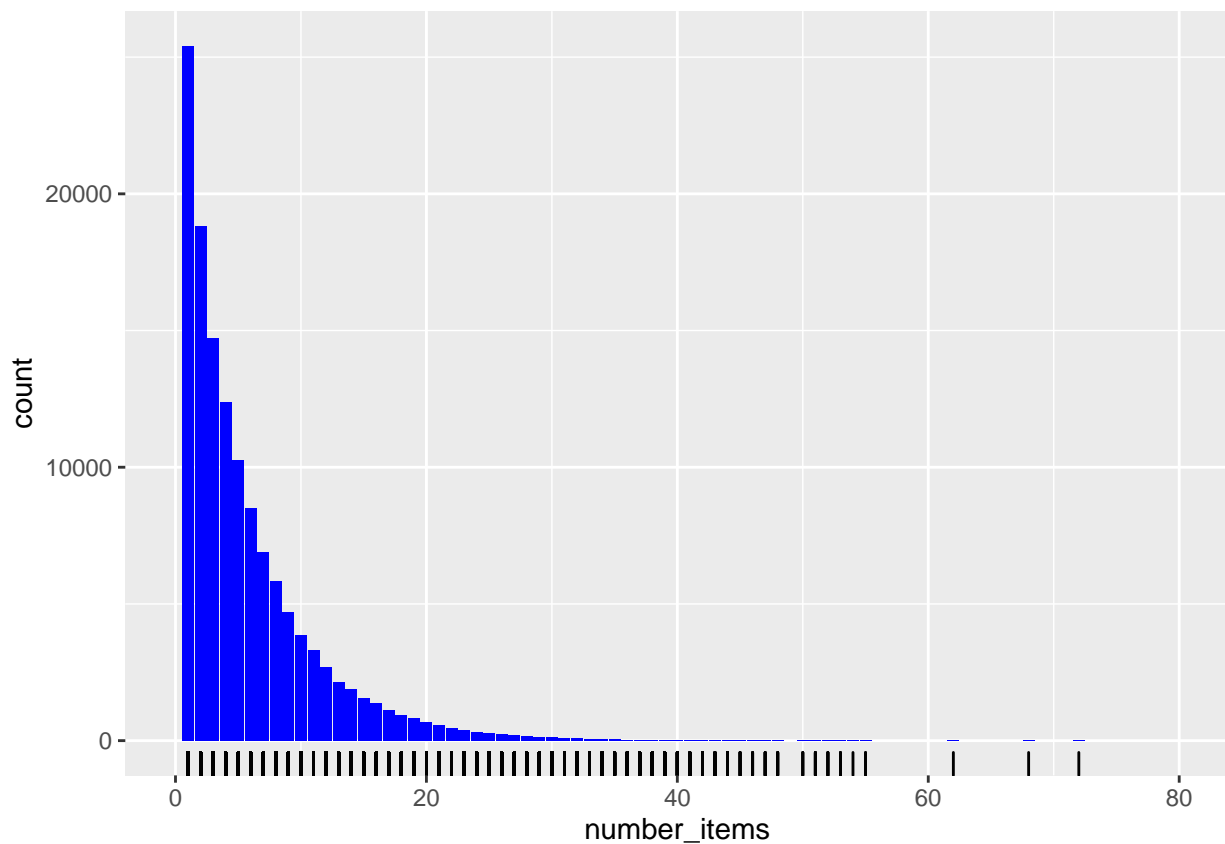
Top 10 products sold and the percentage of transactions they are involved in. Bananas make up 1.35% of the transactions, while organic bananas make up 1.1% of the transactions and organic strawberries make up

0.79% of the transactions.

How many items are in each transaction?

```
X %>%  
  group_by(order_id) %>%  
  dplyr::summarise(number_items=last(add_to_cart_order)) %>%  
  ggplot(aes(x=number_items)) +  
  geom_histogram(stat="count", fill="blue") +  
  geom_rug()+  
  coord_cartesian(xlim=c(0,80))
```

## Warning: Ignoring unknown parameters: binwidth, bins, pad



Items most often reordered:

```
reordered <- X %>%  
  group_by(product_name) %>%  
  dplyr::summarize(proportion_reordered = mean(reordered), n=n()) %>%  
  filter(n>40) %>%  
  top_n(10, wt=proportion_reordered) %>%  
  arrange(desc(proportion_reordered))  
  
kable(reordered)
```

product_name	proportion_reordered	n
2% Lactose Free Milk	0.9347826	92
Organic Low Fat Milk	0.9130435	368
100% Florida Orange Juice	0.8983051	59
Organic Spelt Tortillas	0.8888889	81
Original Sparkling Seltzer Water Cans	0.8888889	45
Banana	0.8841717	18726
Petit Suisse Fruit	0.8833333	120
Organic Lowfat 1% Milk	0.8819876	483
Organic Lactose Free 1% Lowfat Milk	0.8810409	269
1% Lowfat Milk	0.8785249	461

X\_merge column is not needed for association rule mining, so can set to NULL:

```
X$X_merge <- NULL
```

Need to convert dataframe to transaction data so that all items bought together in one order is in one row. Currently different products from the same order are in their own rows (singles format).

```
library(plyr)
```

```
## -----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## -----

##
## Attaching package: 'plyr'

## The following object is masked from 'package:purrr':
##
## compact

## The following objects are masked from 'package:dplyr':
##
## arrange, count, desc, failwith, id, mutate, rename, summarise,
## summarize
```

```
#ddply(dataframe, variables_to_be_used_to_split_data_frame, function_to_be_applied)
```

```
transactionData <- ddply(X, c("order_id","user_id"),
  function(df1)paste(df1$product_name,      #paste is used to concatenate vectors t
    collapse = ","))      #collapse is used to separate the concatenated product name
```

Look at the transaction data. This is currently in the form of a basket format:

*#set order id and user id to NULL in the transaction dataset since it will not be needed for item association*



```
transactionData$order_id <- NULL
transactionData$user_id <- NULL
```

rename column to items

```
colnames(transactionData) <- c("items")
```

write the transaction data csv into a csv file:

```
#write.csv(transactionData, "C:/Users/qt09n/Desktop/Project/market_basket_transactions.csv", quote = FALSE)
```

take the transaction data file which is in basket format and convert it to an object of the transaction class

```
tr
```

```
## transactions in sparse format with
## 131210 transactions (rows) and
## 50153 items (columns)
```

```
summary(tr)
```

```
## transactions as itemMatrix in sparse format with
## 131210 rows (elements/itemsets/transactions) and
## 50153 columns (items) and a density of 0.00020449
##
```

```
## most frequent items:
```

```
##           Banana Bag of Organic Bananas   Organic Strawberries
##           17724                        14597                    10260
## Organic Baby Spinach           Large Lemon           (Other)
##           9318                        7740                    1286023
##
```

```
## element (itemset/transaction) length distribution:
```

```
## sizes
```

```
##      1      2      3      4      5      6      7      8      9     10     11     12     13     14     15     16
## 7750 8047 8474 8470 8887 8684 8471 7856 7104 6447 5943 5356 4770 4219 3645 3364
##  17    18    19    20    21    22    23    24    25    26    27    28    29    30    31    32
## 3073 2617 2390 2020 1771 1560 1408 1245 1085  932   812   637   557   553   434   387
##  33    34    35    36    37    38    39    40    41    42    43    44    45    46    47    48
##  299  289  246  190  178  151  126  116   84   82   71   63   49   43   35   28
##  49    50    51    52    53    54    55    56    57    58    59    60    61    62    63    65
##  21    22    20    23    17    13     8    11     5     5     5     8     5     3     3     1
##  66    67    68    69    72    75    76    78    80    82    84
##   4     3     2     5     1     1     1     2     1     1     1
##
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00    5.00    8.00   10.26   14.00   84.00
##
```

```
## includes extended item information - examples:
```

```
##              labels
## 1              #2
## 2              #2 Coffee Filters
## 3 #2 Cone White Coffee Filters
```

131210 transactions (rows) and 50153 items (columns). 50153 is the product names. Density is the percentage of non-zero cells in a sparse matrix, which is the total number of items purchased divided by a possible number of items in that matrix.

To calculate how many items were purchased:  $131210 \times 50153 \times 0.00020449 = 1345662$

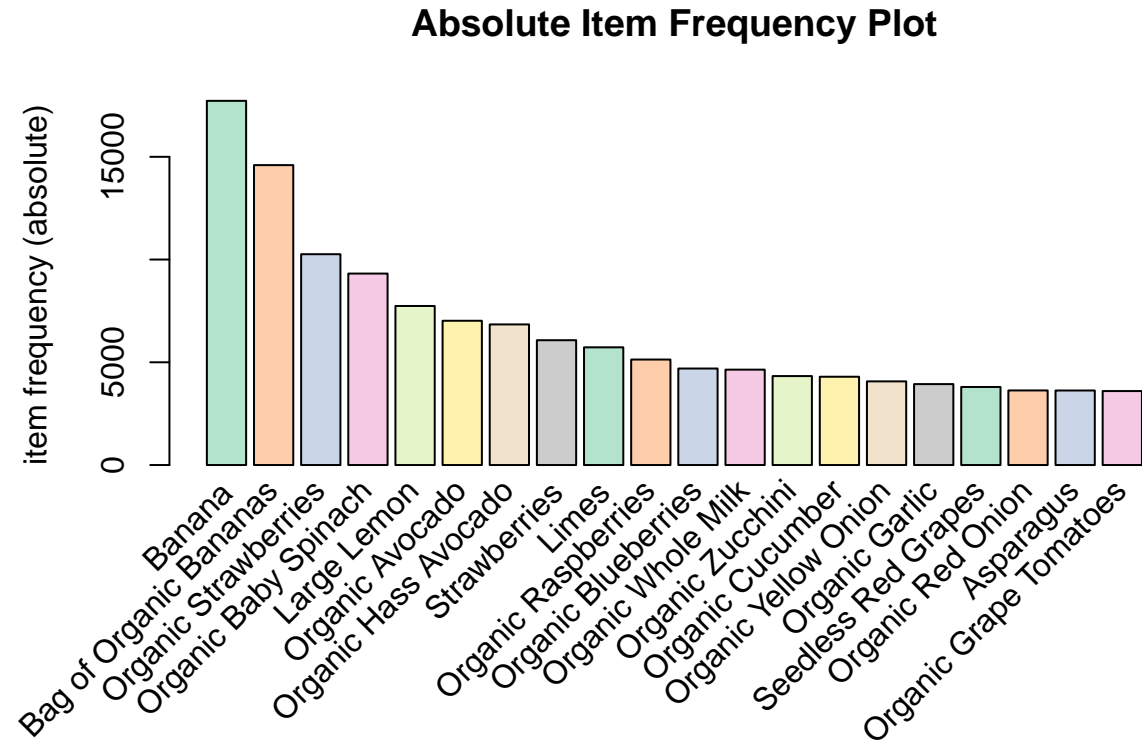
A sparse matrix is a matrix in which most elements are zero.

Element (itemset/transaction) length distribution. This section is about how many transactions containing a certain number of items. The first row is the number of items in a transaction, and the second row is the number of transactions with that number of items. ie. There are 1877502617 transactions with only 1 item. There are 1980472390 transactions with 2 items.

To generate an item Frequency Plot to view the distribution of objects based on itemMatrix.

Create an item frequency plot for the top 50 items.

```
itemFrequencyPlot(tr, topN=20, type="absolute", col=brewer.pal(8, 'Pastel12'), main="Absolute Item Frequency Plot")
```

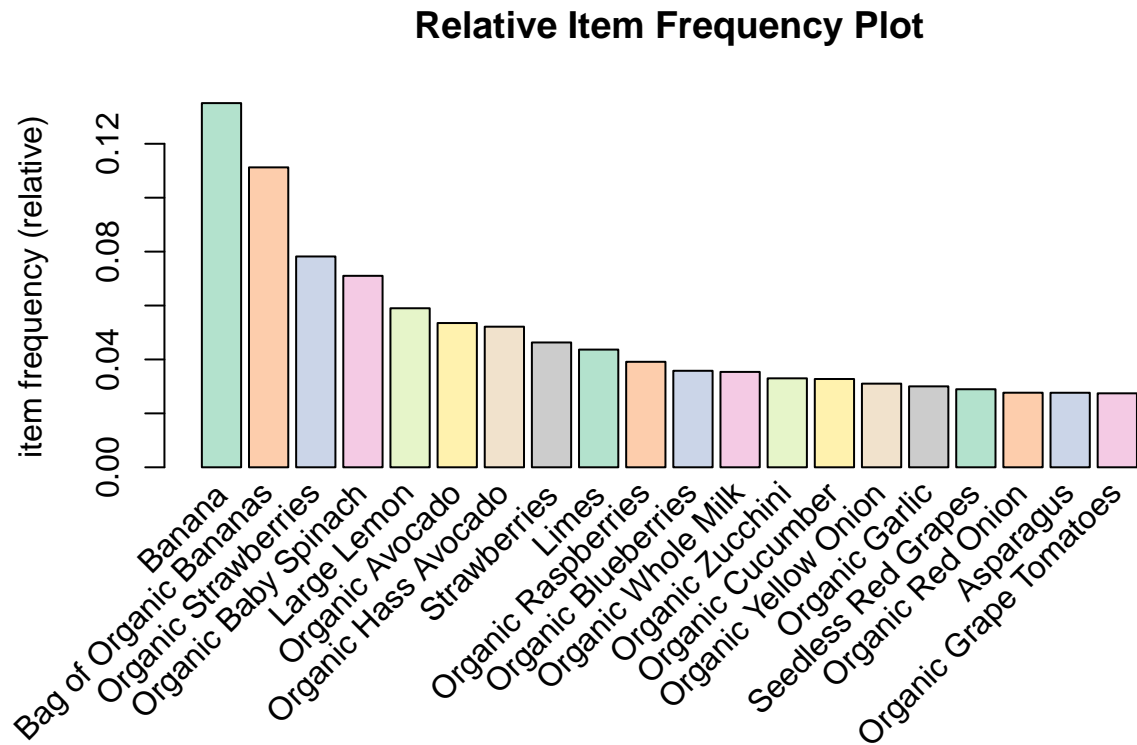


According to the frequency plot, the top 20 products bought in Instacart are banana, bag of organic bananas, organic strawberries, organic baby spinach, large lemon, organic avocado, organic hass avocado, strawberries, limes, organic raspberries, organic blueberries, organic whole milk, organic zucchini, organic cucumber, organic yellow onion, organic garlic, seedless red grapes, organic red onion, asparagus and organic grape tomatoes.

This plot shows absolute frequency which are independent numeric frequencies for each item.

To look at relative frequencies (how many times an item appears in comparison to others):

```
itemFrequencyPlot(tr, topN=20, type="relative", col=brewer.pal(8, 'Pastel12'), main="Relative Item Frequ
```



```
### Generating Rules
```

Mine the rules using APRIORI algorithm.

```
association.rules <- apriori(tr, parameter= list(supp=0.001, conf=0.8, maxlen=10))
```

```
## Apriori
```

```
##
```

```
## Parameter specification:
```

```
## confidence minval smax arem aval originalSupport maxtime support minlen
##          0.8    0.1    1 none FALSE                TRUE         5   0.001    1
```

```
## maxlen target ext
```

```
##          10  rules TRUE
```

```
##
```

```
## Algorithmic control:
```

```
## filter tree heap memopt load sort verbose
```

```
##      0.1 TRUE TRUE  FALSE TRUE    2    TRUE
```

```
##
```

```
## Absolute minimum support count: 131
```

```
##
```

```
## set item appearances ...[0 item(s)] done [0.00s].
```

```
## set transactions ...[50153 item(s), 131210 transaction(s)] done [0.72s].
```

```
## sorting and recoding items ... [1812 item(s)] done [0.02s].
```

```
## creating transaction tree ... done [0.06s].
```

```
## checking subsets of size 1 2 3 4 done [0.05s].
## writing ... [255 rule(s)] done [0.00s].
## creating S4 object ... done [0.01s].
```

The apriori will take tr as the transaction object to apply the rule mining. Parameters allow you to set min\_sup and min\_confidence and min confidence of 0.8, maximum of 10 items(maxlen).

```
summary(association.rules)
```

```
## set of 255 rules
##
## rule length distribution (lhs + rhs):sizes
##   2   3   4
## 132 111  12
##
##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  2.000  2.000   2.000   2.529   3.000   4.000
##
## summary of quality measures:
##      support      confidence      coverage      lift
##  Min.   :0.001021  Min.   :0.8017  Min.   :0.001021  Min.   : 39.52
## 1st Qu.:0.001143 1st Qu.:0.9917 1st Qu.:0.001174 1st Qu.:185.06
## Median :0.001296 Median :1.0000 Median :0.001364 Median :394.02
## Mean   :0.001892 Mean   :0.9800 Mean   :0.001940 Mean   :438.96
## 3rd Qu.:0.002043 3rd Qu.:1.0000 3rd Qu.:0.002058 3rd Qu.:701.98
## Max.   :0.007522 Max.   :1.0000 Max.   :0.007522 Max.   :979.18
##      count
##  Min.   :134.0
## 1st Qu.:150.0
## Median :170.0
## Mean   :248.3
## 3rd Qu.:268.0
## Max.   :987.0
##
## mining info:
## data ntransactions support confidence
##   tr      131210  0.001      0.8
```

set of 255 rules were generated from the apriori algorithm.

to look at just the top 10 rules:

```
inspect(association.rules[1:10])
```

```
##      lhs      rhs      support      confidence
## [1] {Mini & Mobile} => {Natural Artesian Water} 0.001036506 1
## [2] {Americano}    => {Prosciutto}           0.001021264 1
## [3] {1000 Sheet Rolls} => {1â??Ply}           0.001036506 1
## [4] {1â??Ply}      => {1000 Sheet Rolls}      0.001036506 1
## [5] {1000 Sheet Rolls} => {Bathroom Tissue}      0.001036506 1
## [6] {1â??Ply}      => {Bathroom Tissue}      0.001036506 1
## [7] {Twin Pack}     => {French Baguettes}      0.001021264 1
## [8] {French Baguettes} => {Twin Pack}           0.001021264 1
```

```
## [9] {Twin Pack}      => {Take & Bake}      0.001021264 1
## [10] {Take & Bake}     => {Twin Pack}      0.001021264 1
##      coverage    lift      count
## [1] 0.001036506 198.8030 136
## [2] 0.001021264 372.7557 134
## [3] 0.001036506 964.7794 136
## [4] 0.001036506 964.7794 136
## [5] 0.001036506 493.2707 136
## [6] 0.001036506 493.2707 136
## [7] 0.001021264 979.1791 134
## [8] 0.001021264 979.1791 134
## [9] 0.001021264 979.1791 134
## [10] 0.001021264 979.1791 134
```

136 transactions where customers who bought Mini and Mobile also bough Natural Artesian Water. 136 transactions where people who bought 1000 sheet Rolls also bought 1a Ply, and 136 transactions where people who bought 1000 Sheet Rolls also bought Bathroom tissue.

### Limiting the number and size of rules

Setting the the conf value and maxlen parameter to higher values will give stronger rules.

```
shorter_association_rules <- apriori(tr, parameter = list(supp=0.001, conf=0.9, maxlen=5))
```

```
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##      0.9      0.1    1 none FALSE          TRUE      5   0.001      1
## maxlen target  ext
##      5   rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##    0.1 TRUE TRUE  FALSE TRUE     2    TRUE
##
## Absolute minimum support count: 131
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[50153 item(s), 131210 transaction(s)] done [0.72s].
## sorting and recoding items ... [1812 item(s)] done [0.01s].
## creating transaction tree ... done [0.07s].
## checking subsets of size 1 2 3 4 done [0.06s].
## writing ... [231 rule(s)] done [0.00s].
## creating S4 object ... done [0.01s].
```

```
summary(shorter_association_rules)
```

```
## set of 231 rules
##
## rule length distribution (lhs + rhs):sizes
##    2    3    4
```

```

## 121  98  12
##
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.000  2.000  2.000  2.528  3.000  4.000
##
## summary of quality measures:
##      support      confidence      coverage      lift
##  Min.   :0.001021  Min.   :0.9066  Min.   :0.001021  Min.   : 55.65
## 1st Qu.:0.001143  1st Qu.:1.0000  1st Qu.:0.001158  1st Qu.:202.80
## Median :0.001296  Median :1.0000  Median :0.001303  Median :435.52
## Mean   :0.001871  Mean   :0.9932  Mean   :0.001885  Mean   :452.06
## 3rd Qu.:0.002043  3rd Qu.:1.0000  3rd Qu.:0.002050  3rd Qu.:720.96
## Max.   :0.007522  Max.   :1.0000  Max.   :0.007522  Max.   :979.18
##      count
##  Min.   :134.0
## 1st Qu.:150.0
## Median :170.0
## Mean   :245.5
## 3rd Qu.:268.0
## Max.   :987.0
##
## mining info:
## data ntransactions support confidence
##   tr          131210  0.001          0.9

```

```
inspect(shorter_association_rules[1:10])
```

```

##      lhs      rhs      support      confidence
## [1] {Mini & Mobile} => {Natural Artesian Water} 0.001036506 1
## [2] {Americano}    => {Prosciutto}          0.001021264 1
## [3] {1000 Sheet Rolls} => {1â??Ply}          0.001036506 1
## [4] {1â??Ply}      => {1000 Sheet Rolls}      0.001036506 1
## [5] {1000 Sheet Rolls} => {Bathroom Tissue}    0.001036506 1
## [6] {1â??Ply}      => {Bathroom Tissue}    0.001036506 1
## [7] {Twin Pack}     => {French Baguettes}    0.001021264 1
## [8] {French Baguettes} => {Twin Pack}          0.001021264 1
## [9] {Twin Pack}     => {Take & Bake}          0.001021264 1
## [10] {Take & Bake}   => {Twin Pack}          0.001021264 1
##      coverage lift count
## [1] 0.001036506 198.8030 136
## [2] 0.001021264 372.7557 134
## [3] 0.001036506 964.7794 136
## [4] 0.001036506 964.7794 136
## [5] 0.001036506 493.2707 136
## [6] 0.001036506 493.2707 136
## [7] 0.001021264 979.1791 134
## [8] 0.001021264 979.1791 134
## [9] 0.001021264 979.1791 134
## [10] 0.001021264 979.1791 134

```

To remove redundant rules

```
subset.rules <- which(colSums(is.subset(association.rules, association.rules))>1) #get subset rules in
length(subset.rules)
```

```
## [1] 200
```

```
#which() - gives you the position of elements in the vector where value = TRUE
#colSums() - row and column sums for dataframes and numeric arrays
#is.subset() - find out if elements of one vector contain all elements of other vector
```

To remove the subset rules:

```
subset.association.rules <- association.rules[-subset.rules] #remove subset rules
```

To find out what customers buy before buying a certain product, use the appearance option in the apriori command. ie. to find out what people buy before buying French baguettes:

```
baguette.association.rules <- apriori(tr, parameter = list(supp=0.001, conf=0.8), appearance = list(def
```

```
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##          0.8   0.1   1 none FALSE                TRUE     5   0.001     1
## maxlen target  ext
##          10  rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##    0.1 TRUE TRUE  FALSE TRUE     2    TRUE
##
## Absolute minimum support count: 131
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[50153 item(s), 131210 transaction(s)] done [0.74s].
## sorting and recoding items ... [1812 item(s)] done [0.01s].
## creating transaction tree ... done [0.06s].
## checking subsets of size 1 2 3 4 done [0.06s].
## writing ... [3 rule(s)] done [0.01s].
## creating S4 object ... done [0.02s].
```

To find out how many customers buy French baguettes along with other items:

```
inspect(head(baguette.association.rules))
```

```
##      lhs                rhs          support    confidence
## [1] {Take & Bake}      => {French Baguettes} 0.001021264 1
## [2] {Twin Pack}       => {French Baguettes} 0.001021264 1
## [3] {Take & Bake,Twin Pack} => {French Baguettes} 0.001021264 1
## coverage lift count
## [1] 0.001021264 979.1791 134
## [2] 0.001021264 979.1791 134
## [3] 0.001021264 979.1791 134
```

To find out answer to “What other items did customers who bought X item also buy?” ...ie. for French baguettes again:

```
baguette.association.rules <- apriori(tr, parameter = list(supp=0.001, conf=0.8), appearance = list(lhs
```

```
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##          0.8   0.1   1 none FALSE                TRUE     5   0.001     1
## maxlen target  ext
##          10  rules TRUE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##    0.1 TRUE TRUE  FALSE TRUE     2    TRUE
##
## Absolute minimum support count: 131
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[50153 item(s), 131210 transaction(s)] done [0.71s].
## sorting and recoding items ... [1812 item(s)] done [0.02s].
## creating transaction tree ... done [0.05s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [2 rule(s)] done [0.00s].
## creating S4 object ... done [0.03s].
```

Keep lhs as French Baguettes because you want to find out the probability of how many customers buy French baguettes with other items:

```
inspect(head(baguette.association.rules))
```

```
##      lhs                rhs      support    confidence coverage
## [1] {French Baguettes} => {Take & Bake} 0.001021264 1          0.001021264
## [2] {French Baguettes} => {Twin Pack}   0.001021264 1          0.001021264
##      lift      count
## [1] 979.1791 134
## [2] 979.1791 134
```

Scatterplot

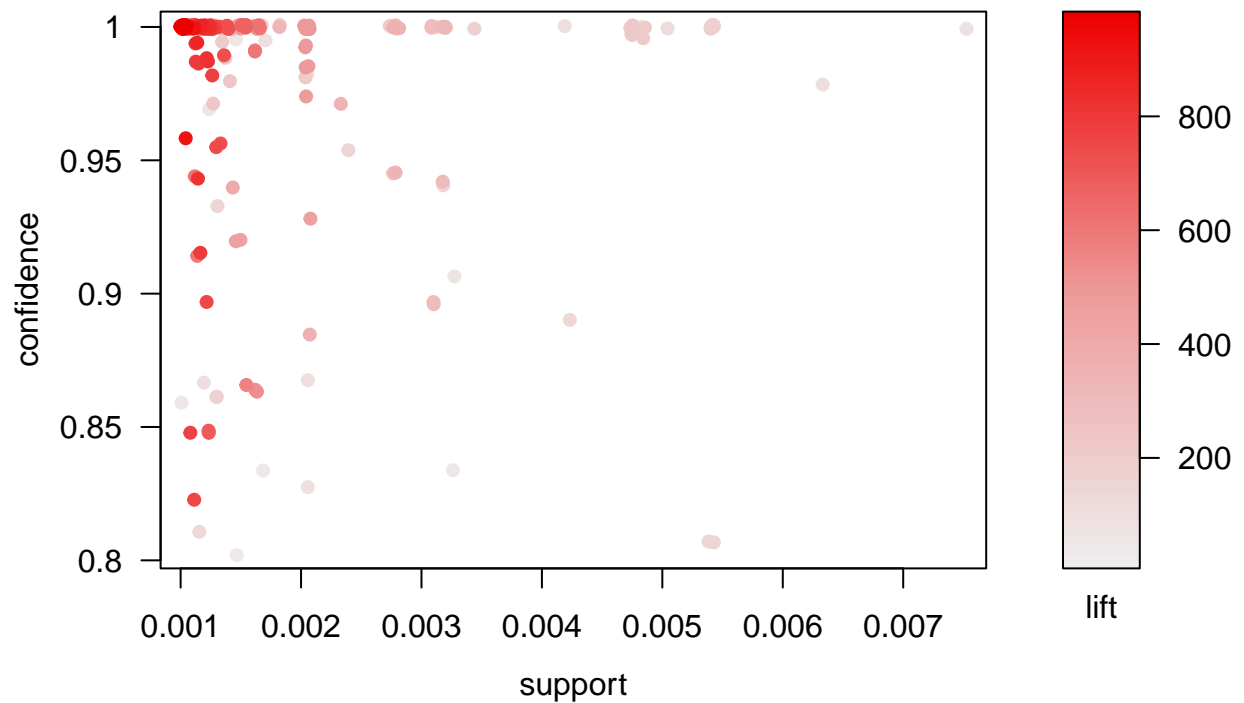
```
#filter rules with confidence greater than 0.6 or 60%
subRules <- association.rules[quality(association.rules)$confidence>0.6]

#plot subrules
plot(subRules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



### Scatter plot for 255 rules

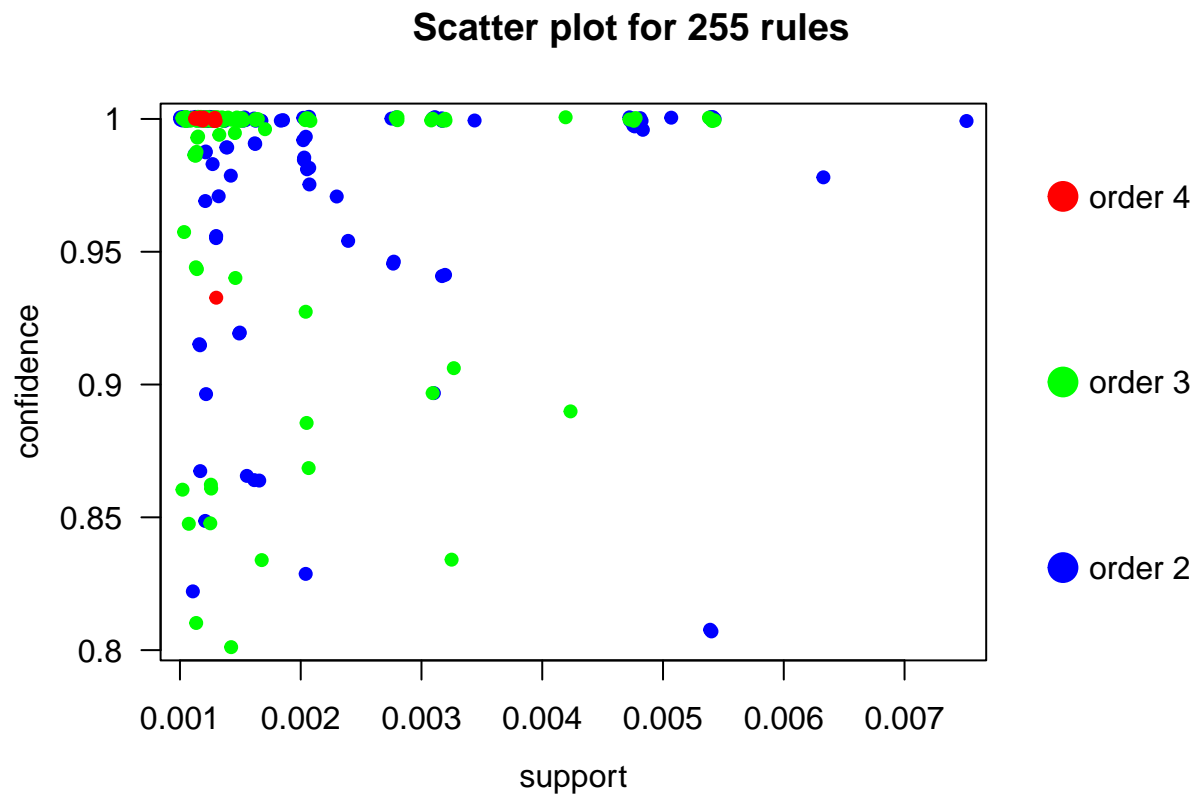


Rules with high lift have low support

Plot options: rulesObject = rules object to be plotted measure= measures for rule interestingness ie. support, confidence, lift or combination of these depending on method value shading = measure used to color points( support, confidence, lift); default=lift metho=visualization method to be used(scatterplot, 2 key plot, matrix3D)

```
plot(subRules, method="two-key plot")
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



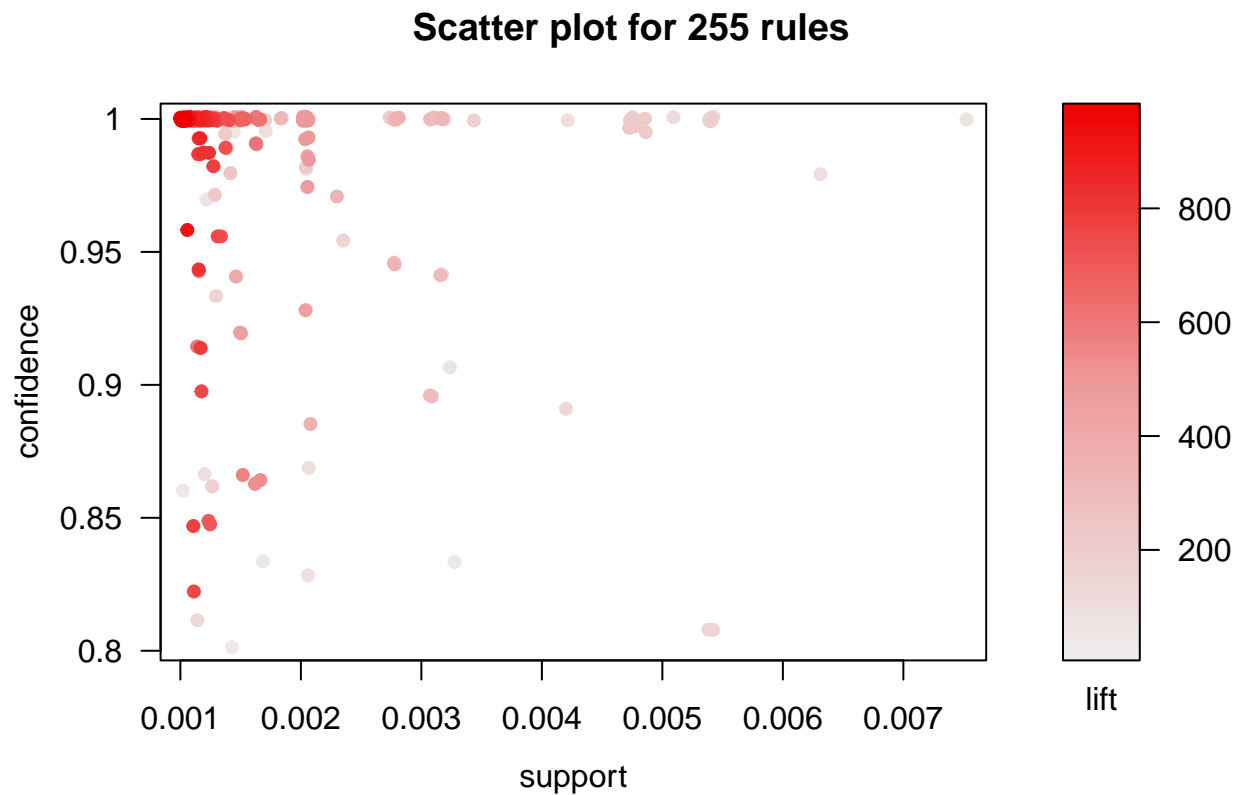
Two key plot has support on x axis and confidence on y-axis. It uses order for coloring. Order is the number of items in the rule.

### Interactive Scatterplot

Users can hover over rules and see the quality measures(support, confidence and lift).

```
plot(subRules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Graph based methods: vertices are labeled with item names; item sets or rules are indicated with a second set of vertices: arrows point from items to rule vertices = LHS; arrow from rule to an item = RHS. Size & color = interest measure.

To get the top 10 rules with highest confidence:

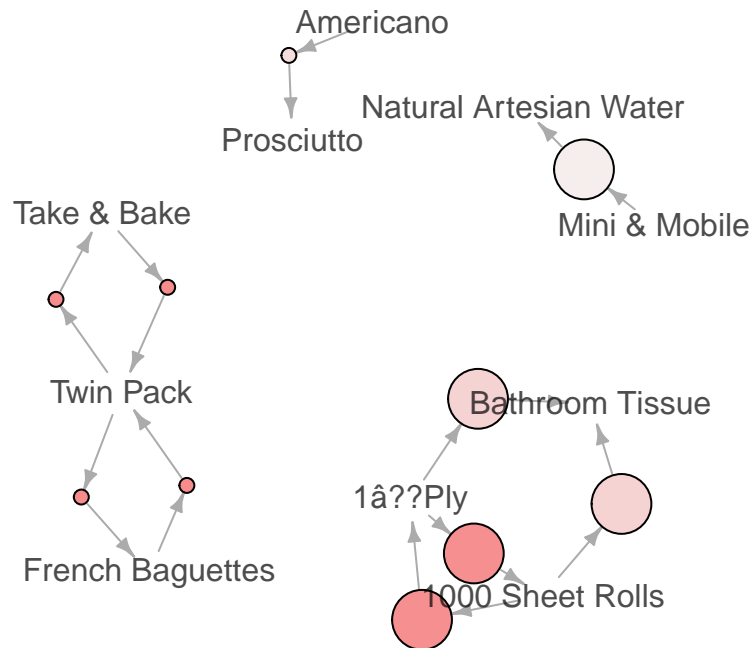
```
top10subRules <- head(subRules, n= 10, by="confidence")
```

Make interactive plot with engine=htmlwidget parameter in plot

```
# plot(top10subRules, method="graph", engine="htmlwidget") #html widget can not be shown in pdf
plot(top10subRules, method="graph")
```

## Graph for 10 rules

size: support (0.001 – 0.001)  
color: lift (198.803 – 979.179)



To export graphs for sets of association rules in GraphML format (which you can open with Gephi tool):

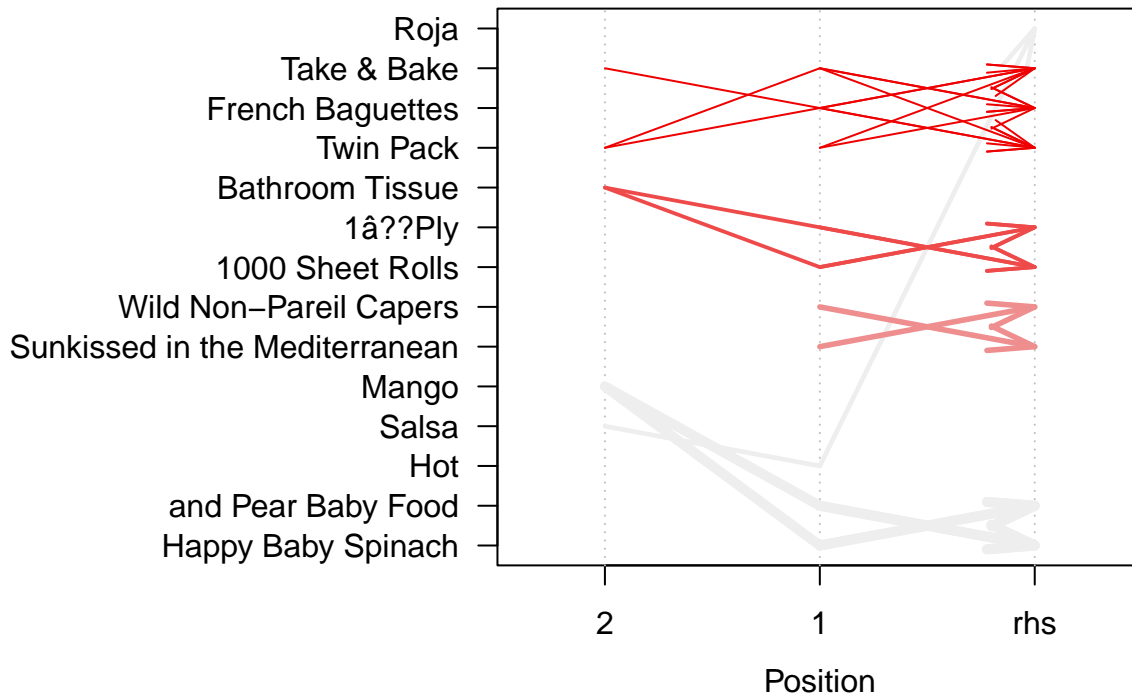
```
saveAsGraph(head(subRules, n=1000, by="lift"), file="rules.graphml")
```

## Individual Rule Representation

This is Parallel Coordinates Plot, used to visualize products with items and types of sales: RHS = consequent, which is item that is suggested for customers to buy: positions are LHS, where 2 = most recent item; and 1=item previously bought

```
#filter top 20 rules with highest lift:
subRules2 <-head(subRules, n=20, by="lift")
plot(subRules2, method="paracoord")
```

## Parallel coordinates plot for 20 rules



According to this plot, if when someone buys salsa, and “hot”.., they are likely to buy Roja.

If someone has mango, and pear baby food in their cart, they are likely to buy Happy Baby Spinach as well.

## Recommender Method 2: Weighted Alternating Least Squares with Implicit Feedback Data

```
#Extract columns for matrix
#transactions<-X[,c("user_id","product_id","order_id")]           #if using training orders data
transactions<-prior_order[,c("user_id","product_id","order_id")] #using prior orders data

#selecting 8K users for modeling
all_users<-unique(transactions$user_id)
randm_users<-sample(all_users,8000L)

#selecting 16K products
all_products<-unique(transactions$product_id)
rand_products<-sample(all_products,16000L)

#final data for matrix
interactions<-transactions %>%
  filter(user_id %in% randm_users & product_id %in% rand_products)
```

```
#find the total orders for each user per product
interactions_sample<-interactions %>%
  group_by(user_id,product_id) %>%
  dplyr::summarise(orders=n())
```

## 'summarise()' has grouped output by 'user\_id'. You can override using the '.groups' argument.

```
dim(interactions)
```

```
## [1] 273611      3
```

```
#encoding users and products
user_enc <- interactions_sample %>%
  distinct(user_id) %>%
  rowid_to_column()

names(user_enc)[names(user_enc) == "rowid"]<- "uid_enc"

product_enc<- interactions_sample %>%
  distinct(product_id) %>%
  rowid_to_column()

names(product_enc)[names(product_enc) == "rowid"]<- "pid_enc"
```

```
#sparse matrix
data<-interactions_sample %>%
  inner_join(user_enc, by = 'user_id') %>%
  inner_join(product_enc, by = 'product_id')

X = sparseMatrix(i = data$uid_enc, j = data$pid_enc, x = data$orders,
  dimnames = list(user_enc$user_id, product_enc$product_id))
```

```
#test set

n_test <- 2000L
test_uid <- sample(nrow(user_enc), n_test)

X_train <- X[-test_uid, ]
X_test <- X[test_uid, ]
```

```
# Split our test set into "history" or "future"
temp = as(X_test, "TsparseMatrix")
temp = data.table(i = temp@i, j = temp@j, x = temp@x)
```

```
temp <- temp %>%
  group_by(i) %>%                                # group by user
  mutate(ct = length(j),                          # number of products each user has)
```

```

    history =
      sample(c(TRUE, FALSE), ct, replace = TRUE, prob = c(.5, .5)) %>%
      select(-ct)

X_test_history <- temp %>% filter(history == TRUE)
X_test_future <- temp %>% filter(history == FALSE)

X_test_history <- sparseMatrix(i = X_test_history$i,
                              j = X_test_history$j,
                              x = X_test_history$x,
                              dims = dim(X_test),
                              dimnames = dimnames(X_test),
                              index1 = FALSE)

X_test_future <- sparseMatrix(i = X_test_future$i,
                              j = X_test_future$j,
                              x = X_test_future$x,
                              dims = dim(X_test),
                              dimnames = dimnames(X_test),
                              index1 = FALSE)

# confidence functions and create matrices

lin_conf <- function(x, alpha) {
  x_confidence <- x
  stopifnot(inherits(x, "sparseMatrix"))
  x_confidence@x = 1 + alpha * x@x
  return(x_confidence)
}

alpha <- .1
lambda <- 10
components <- 10L

#factor matrices for train and test

X_train_conf <- lin_conf(X_train, alpha)
X_test_history_conf <- lin_conf(X_test_history, alpha)

# Initialize a model
# Define hyper parameters
#rank=the number of latent factors in the model (defaults to 10)
#value of alpha is calculated using cross validation

```

```

model <- WRMF$new(rank = components,
                 lambda = lambda,
                 feedback = 'implicit',
                 solver = 'conjugate_gradient')

```

```

# Calculate user factors

```

```

train_user_factors <- model$fit_transform(X_train_conf)

```

```

## INFO [00:03:03.828] starting factorization with 8 threads
## INFO [00:03:05.429] iter 1 loss = 0.6381
## INFO [00:03:06.041] iter 2 loss = 0.3873
## INFO [00:03:06.668] iter 3 loss = 0.3603
## INFO [00:03:07.267] iter 4 loss = 0.3484
## INFO [00:03:08.071] iter 5 loss = 0.3405
## INFO [00:03:08.639] iter 6 loss = 0.3350
## INFO [00:03:09.326] iter 7 loss = 0.3309
## INFO [00:03:09.949] iter 8 loss = 0.3277
## INFO [00:03:10.600] iter 9 loss = 0.3252
## INFO [00:03:11.225] iter 10 loss = 0.3232

```

```

# Products matrix and recommendations are made by selecting the top 10 items for which P(ui) is great

```

```

test_predictions <- model$predict(X_test_history_conf, k = 10)

```

```

#Loss and Score or fixed product factors

```

```

trace = attr(train_user_factors, "trace")

```

```

ggplot(trace) +

```

```

  geom_line(aes(x = iter, y = value, col = scorer)) +

```

```

  labs(title = "Loss and Scoring Metrics by iteration") +

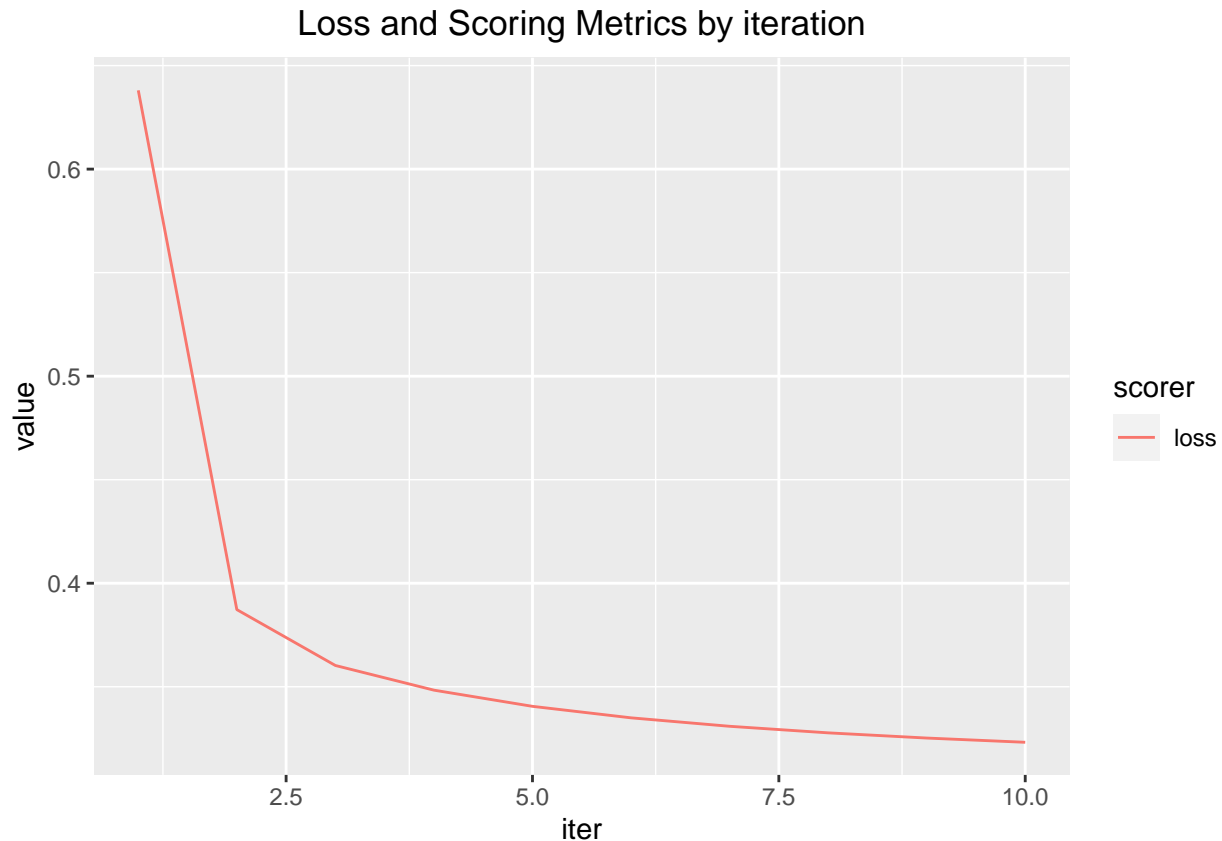
```

```

  theme(plot.title = element_text(hjust = .5))

```





## Recommenderlab for Evaluation of Different Recommender Algorithms in R

Using Recommenderlab in R: 2 types of rating matrix for modelling is available; we will be using the binary rating matrix type where 0 indicates product is not purchased, while 1 indicates product is purchased.

Binary rating matrix is useful when no actual user ratings is available, and it also does not require normalisation.

The rating matrix must be rearranged with orders in rows and products in columns.

```
X <- read.csv("C:/Users/qt09n/Desktop/Project/orders_TRAIN_products_MERGED.csv")
```

```
# train users for top 50
```

```
train_users <- X %>%
  filter(product_id %in% top_products_train$product_id) %>%
  group_by(user_id, product_id, .groups='keep') %>%
  dplyr::summarise(tot=n()) %>%
  ungroup() %>%
  group_by(user_id) %>%
  dplyr::summarise(count2=n()) %>%
  arrange(desc(count2))
```

## 'summarise()' has grouped output by 'user\_id', 'product\_id'. You can override using the '.groups' arg

```
View(train_users)
```

#transaction for top 50 products (train)

```
train_top_50 <- X %>%  
filter(user_id %in%train_users$user_id & product_id %in% top_products_train$product_id)
```

```
dim(train_top_50)
```

```
## [1] 225084      16
```

Check if products are ordered multiple times within the same transaction:

```
retail <- train_top_50 %>%  
  #create a unique identifier for each product in a transaction using the order id and product name info  
  mutate(orderID_product = paste(order_id, product_name, sep=' '))  
#225084 entries
```

```
#filter out duplicates and drop unique identifier  
retail <- retail[!duplicated(retail$orderID_product), ] %>%  
  select(-orderID_product)  
#still 225084 entries, so customers generally do not buy multiples of a single product within the same
```

```
ratings_matrix <- retail %>%  
  select(order_id, product_name) %>%  
  mutate(value=1) %>% #add a column of 1s  
  spread(product_name, value, fill=0) %>% #spread into user-item format  
  select(-order_id) %>%  
  as.matrix() %>%  
  as("binaryRatingMatrix") #convert to recommenderlab class binary matrix  
ratings_matrix
```

```
## 81751 x 50 rating matrix of class 'binaryRatingMatrix' with 225084 ratings.
```

## Evaluation scheme and Model validation

Evaluate the model's effectiveness using recommenderlab's evaluation schemes.

Split the data into a training set and test set with train taking 80% of the data and test taking 20% of the data.

Set method="cross" and k=5 for 5 fold cross-validation. Data will be split into k subsets of equal size, and 80% of data will be used for training and last 20% for evaluation. Models are then estimated recursively 5 times, and a different train/test split is used each time. Results are then averaged to produce a single evaluation set.

```
scheme <- ratings_matrix %>%  
  evaluationScheme(method="cross",  
    k = 5,  
    train = 0.8,
```

```

        given = -1)    #selecting given = -1 means that for the test users 'all but 1' random
scheme

```

```

## Evaluation scheme using all-but-1 items
## Method: 'cross-validation' with 5 run(s).
## Good ratings: NA
## Data set: 81751 x 50 rating matrix of class 'binaryRatingMatrix' with 225084 ratings.

```

## Set up a list of algorithms

Create a list of algorithms from recommenderlab and specify model parameters. Consider schemes which evaluate on the binary rating matrix and include random items algorithm for benchmarking.

```

algorithms <- list(
  "association rules" = list(name = "AR",
                             param = list(supp=0.001, conf=0.01)),
  "random items" = list(name = "RANDOM", param=NULL),           #randomly chosen items for comparison
  "popular items" = list(name = "POPULAR", param = NULL),      #recommender based on item popularity
  "item-based CF" = list(name = "IBCF", param = list(k=5))     #recommender based on item-based CF
  "user-based CF" = list(name = "UBCF",                        #recommender based on user-based CF
    #                    param = list(method = "Cosine", nn= 50)) #user-based CF often runs into memory
  )

```

Pass the scheme and algorithms to the evaluate() function, to evaluate several recommender algorithms using an evaluation scheme. The end product is a evaluation result list.

elect type= topNList to evaluate a Top N List of product recommendations and specify how many recommendations to calculate with the parameter n = c(1,3,5,10,15,20)

Note: for the UBCF method, it is important to allocate enough memory in RStudio for processing.

To check the current limit in R session use memory.limit(); and then to increase the size of memory use memory.limit(size=n) ie. memory.limit(size=56000)

```

#run garbage collection to free up memory for analysis:
gc()

```

```

##           used   (Mb) gc trigger   (Mb) max used   (Mb)
## Ncells  4076762 217.8  11370580 607.3  11370580 607.3
## Vcells 368769230 2813.5 719432968 5488.9 719196359 5487.1

```

```

#remove from global environment variables which are not needed for the analysis:
# rm(X, transactionData, train_users, tr, top10subRules, top_products_train, subset.association.rules, ...)

#change memory limit to 56000 Mb to prevent problems with memory limit ("cannot allocate vector of size 56000")
memory.limit(size=56000)

```

```

## [1] 56000

```

```
results <- recommenderlab::evaluate(scheme,
                                   algorithms,
                                   type= "topNList",
                                   n = c(1,3,5,10, 15, 20))
```

```
## AR run fold/sample [model time/prediction time]
## 1 [0.04sec/201.11sec]
## 2 [0.05sec/245.57sec]
## 3 [0.07sec/248.34sec]
## 4 [0.06sec/192.39sec]
## 5 [0.07sec/207.48sec]
## RANDOM run fold/sample [model time/prediction time]
## 1 [0sec/2.25sec]
## 2 [0sec/1.95sec]
## 3 [0sec/1.96sec]
## 4 [0sec/1.82sec]
## 5 [0sec/1.91sec]
## POPULAR run fold/sample [model time/prediction time]
## 1 [0sec/12.79sec]
## 2 [0sec/11.86sec]
## 3 [0sec/11.64sec]
## 4 [0.02sec/11.72sec]
## 5 [0.01sec/11.98sec]
## IBCF run fold/sample [model time/prediction time]
## 1 [0.22sec/1.52sec]
## 2 [0.35sec/1.42sec]
## 3 [0.25sec/1.52sec]
## 4 [0.24sec/1.72sec]
## 5 [0.25sec/1.54sec]
```

```
results
```

```
## List of evaluation results for 4 recommenders:
##
## $'association rules'
## Evaluation results for 5 folds/samples using method 'AR'.
##
## $'random items'
## Evaluation results for 5 folds/samples using method 'RANDOM'.
##
## $'popular items'
## Evaluation results for 5 folds/samples using method 'POPULAR'.
##
## $'item-based CF'
## Evaluation results for 5 folds/samples using method 'IBCF'.
```

```
names(results)
```

```
## [1] "association rules" "random items"      "popular items"
## [4] "item-based CF"
```

```
#Access individual results by list subsetting using an index or the name specified when calling evaluate
results[["item-based CF"]]
```

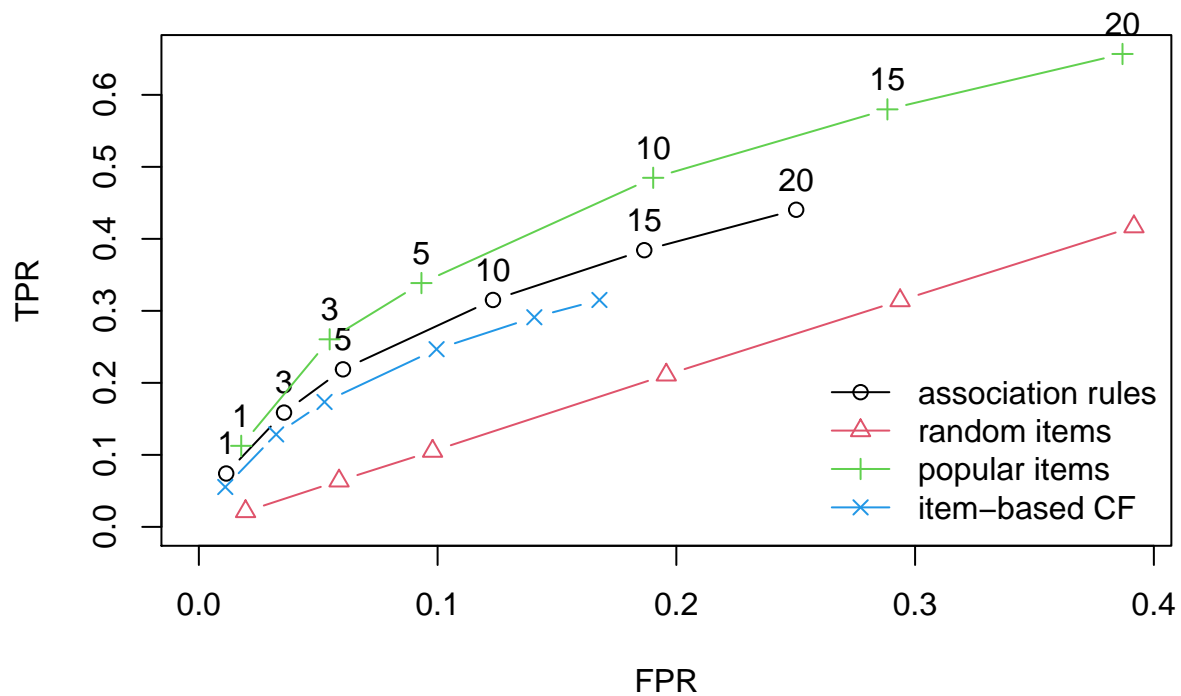
```
## Evaluation results for 5 folds/samples using method 'IBCF'.
```

## Visualise the Results

Use plot function from recommenderlab to compare model performance

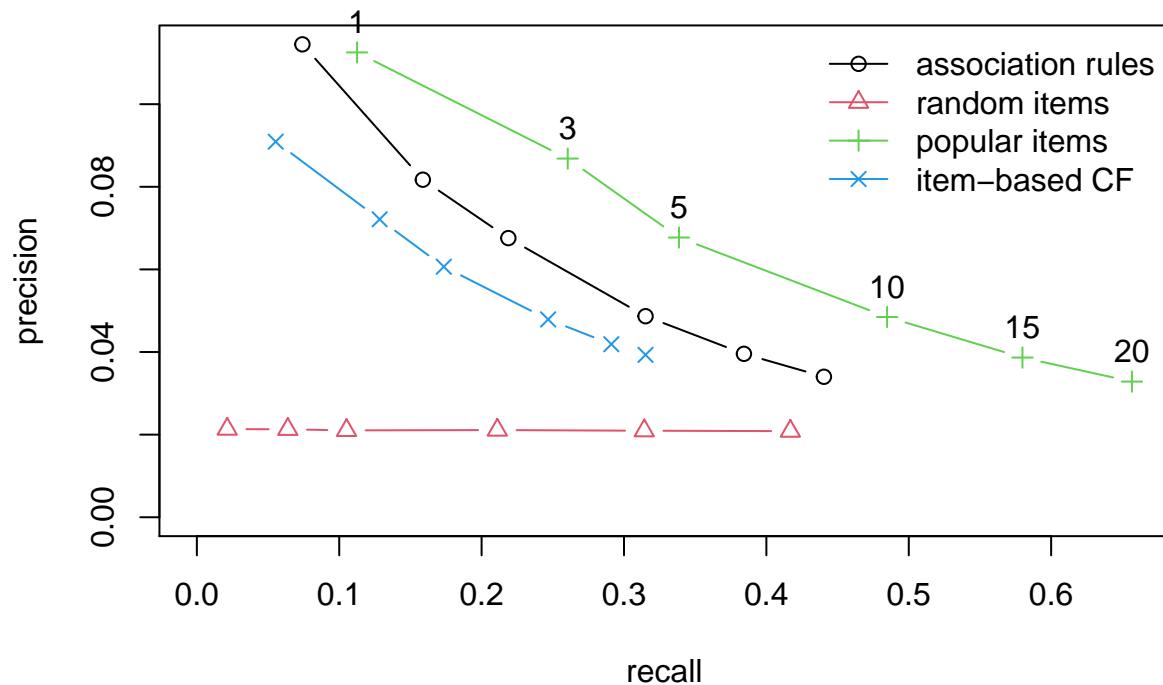
Arrange confusion matrix for one model in a convenient format:

```
plot(results, annotate=c(1,3), legend="bottomright")
```



Comparison of ROC curves. For this dataset and the given evaluation scheme, popular items and association rules outperform the other methods, in providing a better combination of TPR and FPR amongst the 4 algorithms evaluated for the top-N list recommendations.

```
plot(results, "prec/rec", annotate=3, legend="topright")
```



Comparison of precision vs. recall curves for the 4 recommender algorithms shows that Popular items and association rules performed the best for the given evaluation scheme.

## References

<https://towardsdatascience.com/market-basket-analysis-with-recommenderlab-5e8bdc0de236>

Michael Hahsler (2021). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-7. <https://github.com/mhahsler/recommenderlab>

<https://www.datacamp.com/community/tutorials/market-basket-analysis-r#code>

<https://datascienceplus.com/a-gentle-introduction-on-market-basket-analysis%E2%80%8A-%E2%80%8Aassociation-rules/>

[https://en.wikipedia.org/wiki/Sparse\\_matrix](https://en.wikipedia.org/wiki/Sparse_matrix)

<https://cran.r-project.org/web/packages/arulesViz/vignettes/arulesViz.pdf>

<https://github.com/willsmorgan/Recommender-Systems-using-W-ALS/blob/master/W-ALS%20Final.pdf>