OneMount Data test

Khiem Phung

9/28/2021

Install all required packages

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5 v purrr 0.3.4  
## v tibble 3.1.4 v stringr 1.4.0  
## v tidyr 1.1.3 v forcats 0.5.1  
## v readr 2.0.1

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(readxl)  
library(lubridate)

##   
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':  
##   
## date, intersect, setdiff, union

Then we import the data set

#import the data  
df <- read\_excel("C:/Users/khiem.phung/Downloads/Test\_Data\_Skill.xlsx",   
 sheet = "Data")

SQL Test

1.

###SQL test  
#first two services and the date   
df\_first2 <- df %>%  
 select(User\_id,Serviceid,Date) %>%  
 group\_by(User\_id) %>%  
 arrange(Date) %>%  
 group\_by(User\_id) %>%  
 slice(1:2)

#last service and the date  
df\_last <- df %>%  
 select(User\_id,Serviceid,Date) %>%  
 group\_by(User\_id) %>%  
 arrange(desc(Date)) %>%  
 group\_by(User\_id) %>%  
 slice(1)

#distinct serviceid that users use  
df\_service <- df %>%  
 distinct(User\_id,Serviceid,Date) %>%  
 group\_by(User\_id) %>%  
 count()

Present the results in SQL format

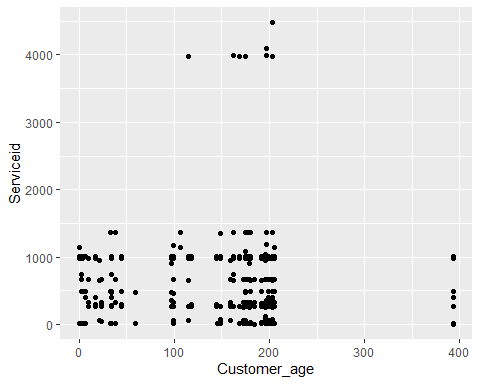
#put data in wide format  
df\_first\_wide <- df\_first2 %>%  
 merge(df\_first2[-1,], by = 'User\_id') %>%  
 group\_by(User\_id) %>%  
 slice(1)  
  
#merge all tables together   
df\_sql <- df\_first\_wide %>%  
 inner\_join(df\_last, by = 'User\_id') %>%  
 inner\_join(df\_service, by = 'User\_id')  
  
#rename the columns   
df\_sql <- df\_sql %>%  
 rename(FirstServiceid = Serviceid.x,  
 FirstServiceDate = Date.x,  
 SecondServiceid = Serviceid.y,  
 SecondServiceDate = Date.y,  
 LastServiceid = Serviceid,  
 LastServiceDate = Date,  
 TotalService = n)  
  
#reorder columns   
df\_sql <- df\_sql[c(1,2,4,3,5,6,7,8)]  
df\_sql

## # A tibble: 44 x 8  
## # Groups: User\_id [44]  
## User\_id FirstServiceid SecondServiceid FirstServiceDate   
## <dbl> <dbl> <dbl> <dttm>   
## 1 2464231 18 667 2018-01-07 00:00:00  
## 2 3676235 273 273 2018-01-05 00:00:00  
## 3 4958104 946 946 2018-01-22 00:00:00  
## 4 11642209 20 20 2018-01-01 00:00:00  
## 5 18246539 984 984 2018-03-22 00:00:00  
## 6 20831230 667 667 2018-01-01 00:00:00  
## 7 21307673 1014 1014 2018-01-03 00:00:00  
## 8 25674126 1136 1136 2018-01-02 00:00:00  
## 9 30456022 481 481 2018-01-03 00:00:00  
## 10 31485023 1014 1014 2018-01-03 00:00:00  
## # ... with 34 more rows, and 4 more variables: SecondServiceDate <dttm>,  
## # LastServiceid <dbl>, LastServiceDate <dttm>, TotalService <int>

Analysis Test

1.

#duplicate to a new df   
df\_sql2 <- df\_sql  
#create customer's age column  
df\_sql2$Customer\_age <- (df\_sql$LastServiceDate  
 - df\_sql$FirstServiceDate)  
df\_sql2$Customer\_age <- time\_length(df\_sql2$Customer\_age,  
 unit = 'days')  
  
#create table with user id and their age   
df\_age <- df\_sql2 %>%  
 select(User\_id, Customer\_age)  
#join that table with the original table to get user id, their age, and all  
#service ids they use  
df\_age <- df\_age %>%  
 inner\_join(df, by = 'User\_id')  
  
#select only age groups and all service id used   
df\_age2 <- df\_age %>%  
 ungroup(User\_id) %>%  
 select(Customer\_age, Serviceid) %>%  
 distinct()  
  
#initially visualize the clusters using scatterplot  
ggplot(df\_age2, aes(x = Customer\_age,  
 y = Serviceid)) +  
 geom\_point()



Based on the plot, we can see that the service ids are clustered in three main groups:

- Customers with age less than 3 months.

- Customers with age from 3 months to a year.

- Customers with age over a year,

We then find all the service ids based on those groups.

* Service ids of customers with age less than 3 months:

#print out serviceid with customer age < 92  
df\_age2 %>%  
 group\_by(Customer\_age) %>%  
 filter(Customer\_age < 92) %>%  
 ungroup(Customer\_age) %>%  
 select(Serviceid)

## # A tibble: 55 x 1  
## Serviceid  
## <dbl>  
## 1 984  
## 2 269  
## 3 666  
## 4 326  
## 5 667  
## 6 11  
## 7 333  
## 8 1366  
## 9 481  
## 10 18  
## # ... with 45 more rows

* Service ids of customers with age between 3 months and a year

#print out serviceid with customer age > 90 and < 365  
df\_age2 %>%  
 group\_by(Customer\_age) %>%  
 filter(Customer\_age > 90, Customer\_age < 365) %>%  
 ungroup(Customer\_age) %>%  
 select(Serviceid)

## # A tibble: 246 x 1  
## Serviceid  
## <dbl>  
## 1 18  
## 2 667  
## 3 333  
## 4 1366  
## 5 268  
## 6 3995  
## 7 4095  
## 8 982  
## 9 363  
## 10 1186  
## # ... with 236 more rows

* Service ids of customers with age over a year

#print out serviceid with customer age > 365  
df\_age2 %>%  
 group\_by(Customer\_age) %>%  
 filter(Customer\_age > 365) %>%  
 ungroup(Customer\_age) %>%  
 select(Serviceid)

## # A tibble: 7 x 1  
## Serviceid  
## <dbl>  
## 1 2  
## 2 19  
## 3 981  
## 4 487  
## 5 1014  
## 6 271  
## 7 398

1. Cross-sales service

Call the required library.

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

library(arules)

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack

##   
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(arulesViz)

First, we have to group all the Service ids customers used by User id and date.

#get transaction data by putting all service ids on one row, grouped by  
#users and the date users used those services  
df\_transaction <- ddply(df,c('User\_id','Date'),  
 function(df1)paste(df1$Serviceid,  
 collapse = ','))  
#select on the service column   
df\_transaction <- df\_transaction %>%  
 select(V1)

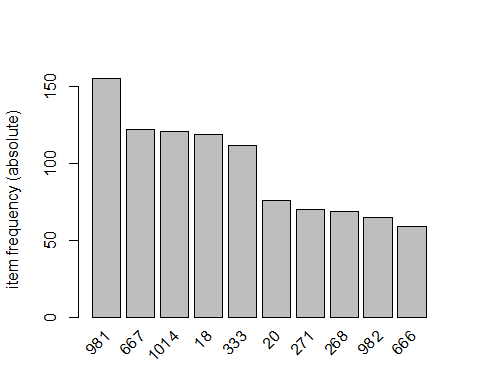
This would be called the basket format. For analysis, we will store this transaction data to .csv file and load it back to convert it into an object of the transaction class.

#store the data to a csv file   
write.csv(df\_transaction,  
'C:/Users/khiem.phung/Downloads/basket\_transaction.csv',  
quote = FALSE, row.names = FALSE)  
  
#load the data into transaction class  
tr <- read.transactions('C:/Users/khiem.phung/Downloads/basket\_transaction.csv',  
 format = 'basket', sep=',')

## Warning in asMethod(object): removing duplicated items in transactions

We can see the top 10 service used by customers.

#see the service with most frequent appearance   
itemFrequencyPlot(tr,topN=10,type="absolute")



This plot shows that Service id of 981, 667, and 1014 were used the most by customers. Thus, one thing that we can do is to focus more on these services for development and analysis in the future.

We then proceed to mine the rules using the ARRIORI algorithm.

#mine the rules using the APRIORI algorithm  
association.rules <- apriori(tr,   
 parameter = list(supp=0.001, conf=0.8))

## 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: 0   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[80 item(s), 563 transaction(s)] done [0.00s].  
## sorting and recoding items ... [80 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 5 6 7 8 done [0.00s].  
## writing ... [2756 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

summary(association.rules)

## set of 2756 rules  
##   
## rule length distribution (lhs + rhs):sizes  
## 2 3 4 5 6 7 8   
## 76 570 946 754 326 77 7   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2.000 4.000 4.000 4.342 5.000 8.000   
##   
## summary of quality measures:  
## support confidence coverage lift   
## Min. :0.001776 Min. :0.8000 Min. :0.001776 Min. : 2.906   
## 1st Qu.:0.001776 1st Qu.:1.0000 1st Qu.:0.001776 1st Qu.: 4.653   
## Median :0.001776 Median :1.0000 Median :0.001776 Median : 8.043   
## Mean :0.003264 Mean :0.9978 Mean :0.003312 Mean : 32.066   
## 3rd Qu.:0.001776 3rd Qu.:1.0000 3rd Qu.:0.001776 3rd Qu.: 9.877   
## Max. :0.198934 Max. :1.0000 Max. :0.216696 Max. :563.000   
## count   
## Min. : 1.000   
## 1st Qu.: 1.000   
## Median : 1.000   
## Mean : 1.837   
## 3rd Qu.: 1.000   
## Max. :112.000   
##   
## mining info:  
## data ntransactions support confidence  
## tr 563 0.001 0.8

#sort the rules by count   
association.rules <- sort(association.rules, by="count", decreasing=TRUE)

The apriori will take tr as the transaction object on which mining is to be applied. parameter will allow you to set min\_sup and min\_confidence. The default values for parameter are minimum support of 0.1, the minimum confidence of 0.8.

Since there is a total of 2756 rules, we will look at the top 10 rules.

#print top 10 rules  
inspect(association.rules[1:10])

## lhs rhs support confidence coverage lift count  
## [1] {333} => {667} 0.19893428 1.0000000 0.19893428 4.614754 112   
## [2] {667} => {333} 0.19893428 0.9180328 0.21669627 4.614754 112   
## [3] {982} => {268} 0.11367673 0.9846154 0.11545293 8.033891 64   
## [4] {268} => {982} 0.11367673 0.9275362 0.12255773 8.033891 64   
## [5] {326} => {666} 0.10124334 1.0000000 0.10124334 9.542373 57   
## [6] {666} => {326} 0.10124334 0.9661017 0.10479574 9.542373 57   
## [7] {271} => {981} 0.10124334 0.8142857 0.12433393 2.957696 57   
## [8] {18,982} => {268} 0.07460036 0.9767442 0.07637655 7.969666 42   
## [9] {18,268} => {982} 0.07460036 1.0000000 0.07460036 8.661538 42   
## [10] {13,333} => {667} 0.07282416 1.0000000 0.07282416 4.614754 41

Using the above output, we can identify the cross sales items as:

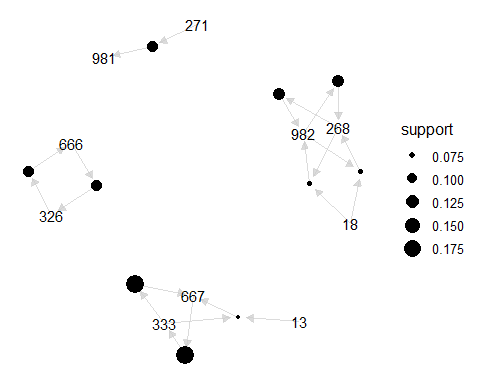
- 100% of customers who used service 333 would also use service 667, based on 112 counts.

- 98% of customers who used service 982 would also use service 268, based on 64 counts.

- 100% of customers who used service 326 would also use service 666, based on 57 counts.

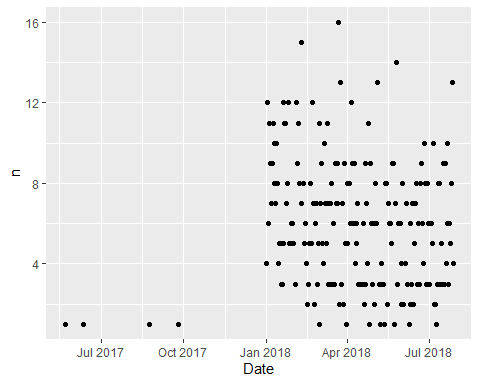
We could present our analysis in a visually appealing manner by drawing a chart of connected items.

#visualize the rules   
plot(association.rules[1:10],method="graph",   
 shading = NA)



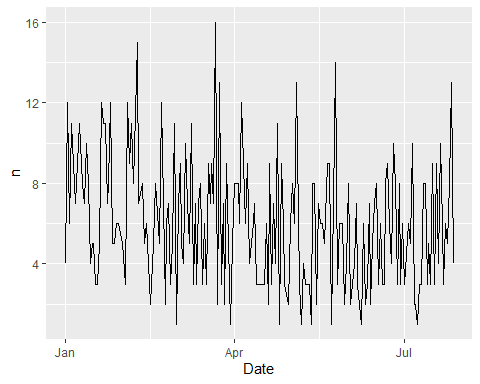
Bonus question: One thing that we as service providers could analyze is when the customers were using the services, and by how much. We then could count total number of distinct services used each date.

##Bonus question  
detach("package:plyr", unload=TRUE)  
#count number of service ids used each date   
df\_date <- df %>%  
 distinct(Date, Serviceid) %>%  
 group\_by(Date) %>%  
 count()  
  
#visualize the findings  
ggplot(df\_date, aes(x = Date, y = n)) +  
 geom\_point()



We noticed that most of the services are used in 2018. Thus, we can remove those used in 2017 for a better view.

#filter the date to 2018  
df\_date\_2018 <- df\_date %>%  
 filter(Date > '2017-12-31')  
  
#visualize the new series   
ggplot(df\_date\_2018, aes(x = Date, y = n)) +  
 geom\_line()



Now that we have a time series of all services used by customers from Jan 2018 to July 2018. With this, we can analyze whether or not this has seasonality, as well as the trend of services used if any. From that, we can also predict the number of services used in upcoming days.