Market Basket Analysis

Shrinath

March 1, 2018

Description: Recommender system using the concept of Market basket analysis. We have used Apriori Algorithm to predict top 20 most sold items and relevant items related to highest confidence. Expected growth in purchased rate is 14%.

```
#install.packages("RColorBrewer")
#install.packages("arulesViz", dependencies = TRUE)
library("devtools")
install_github("mhahsler/arulesViz")
```

```
## Skipping install of 'arulesViz' from a github remote, the SHA1 (4b9aa693) has not changed since last install.
##   Use `force = TRUE` to force installation
```

```
library(arulesViz)
```

```
## Loading required package: arules
```

```
## Warning: package 'arules' was built under R version 3.4.4
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'
```

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

```
## Loading required package: grid
```

```
library(RColorBrewer)
library(arules)
dataset = read.csv('Market_Basket_Optimisation.csv', header = FALSE)
head(dataset)
```

```
##               V1        V2         V3               V4           V5
## 1         shrimp   almonds    avocado   vegetables mix green grapes
## 2        burgers meatballs       eggs
## 3        chutney
## 4         turkey   avocado
## 5  mineral water      milk energy bar whole wheat rice    green tea
## 6 low fat yogurt
##                 V6   V7             V8           V9          V10
## 1 whole weat flour yams cottage cheese energy drink tomato juice
## 2
## 3
## 4
## 5
## 6
##              V11       V12   V13   V14           V15    V16
## 1 low fat yogurt green tea honey salad mineral water salmon
## 2
## 3
## 4
## 5
## 6
##                 V17             V18     V19       V20
## 1 antioxydant juice frozen smoothie spinach olive oil
## 2
## 3
## 4
## 5
## 6
```

```
View(dataset)
```

Description: This dataset contains 20 variables with 7500 observations.7500 customers purchase history on weekly basis.But we are not going to use this dataset because Avril's package doesn't take dataset like this as input.It takes input as the sparse matrix.

```
dataset = read.transactions('Market_Basket_Optimisation.csv', sep = ',', rm.duplicates = TRUE)
```

```
## distribution of transactions with duplicates:
## 1
## 5
```

```
#There are 5 transactions containing 1 duplicates
str(dataset)
```

```
## Formal class 'transactions' [package "arules"] with 3 slots
##   ..@ data       :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
##   .. .. ..@ i       : int [1:29358] 0 1 3 32 38 47 52 53 59 64 ...
##   .. .. ..@ p       : int [1:7502] 0 20 23 24 26 31 32 34 37 40 ...
##   .. .. ..@ Dim     : int [1:2] 119 7501
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ factors : list()
##   ..@ itemInfo   :'data.frame':  119 obs. of  1 variable:
##   .. ..$ labels: chr [1:119] "almonds" "antioxydant juice" "asparagus" "avocado" ...
##   ..@ itemsetInfo:'data.frame':  0 obs. of  0 variables
```

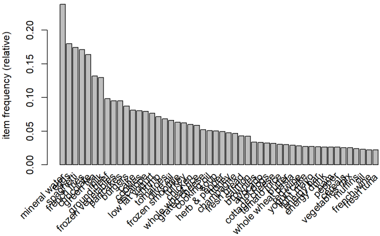
Description: It's actually a matrix that contains a lot of zeroes in machinery and we will encounter a lot of times the word sparcity that corresponds to a large number of zeroes.So this matrix contains very few number of non-zero values.In this 120 different products are present and make 120 columns.Lines will be same as different transactions.So 0 and 1 in the new matrix.0 represent customer has not bought the product and 1 represent customer has bought the product.We need to use sep function because of read.transaction doesn't understand comma separator rm.duplicates is to avoid duplicates.

```
summary(dataset)
```

```
## transactions as itemMatrix in sparse format with
##  7501 rows (elements/itemsets/transactions) and
##  119 columns (items) and a density of 0.03288973
##
## most frequent items:
## mineral water          eggs     spaghetti  french fries     chocolate
##          1788          1348          1306          1282          1229
##       (Other)
##         22405
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17
##   16   18   19   20
##    4    1    2    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   3.914   5.000  20.000
##
## includes extended item information - examples:
##             labels
## 1           almonds
## 2 antioxydant juice
## 3         asparagus
```

we can observe that 7501 rows and 119 columns and a density of 0.03.Density is proportion of non-zero values is 0.03.3% non-zero and 97% zero.Most frequent item is mineral water.Eggs take 2nd place and so on.Length distribution defines itemsets per transaction.1754 basket contains a single item.1358 basket contains two products.Mean is 3.9 and max are 20.

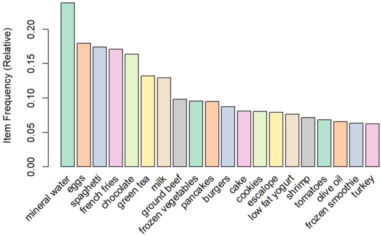
```
itemFrequencyPlot(dataset, topN = 50)
```



Here is a list of top 50 most frequent purchased products

```
itemFrequencyPlot(dataset,topN=20,col=brewer.pal(8,'Pastel2'),main='Relative Item Frequency Plot',type="relative"
,ylab="Item Frequency (Relative)")
```

Relative Item Frequency Plot



Here is a list of top 20 most frequent purchased products

```
# Training Apriori on the dataset
# COnsidering item to be bought 3 times a day that defines support as 0.003 and considering confidence 0.8 by default value
rules = apriori(data = dataset, parameter = list(support = 0.003, confidence = 0.8))
```

```
## Apriori
## 
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.003      1
##  maxlen target   ext
##      10  rules FALSE
## 
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
## 
## Absolute minimum support count: 22 
## 
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [115 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

We can observe that with 0.8 confidence no rules can be generated.

```
# COnsidering item to be bought 3 times a day that defines support as 0.003 and considering confidence 0.4 by default value
#Support 3*7/7500 ~ 0.003
rules = apriori(data = dataset, parameter = list(support = 0.003, confidence = 0.4))
```

```
## Apriori
## 
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5   0.003      1
##  maxlen target   ext
##      10  rules FALSE
## 
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
## 
## Absolute minimum support count: 22 
## 
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [115 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.08s].
## writing ... [281 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
#Inspecitng top 20 rules with support 0.03 and confidence of 40%

inspect(sort(rules, by = 'lift')[1:20])
```

```
##      lhs                    rhs                    support confidence     lift count
## [1]  {mineral water,
##       whole wheat pasta} => {olive oil}         0.003866151  0.4027778 6.115863    29
## [2]  {spaghetti,
##       tomato sauce}      => {ground beef}       0.003066258  0.4893617 4.980600    23
## [3]  {french fries,
##       herb & pepper}     => {ground beef}       0.003199573  0.4615385 4.697422    24
## [4]  {cereals,
##       spaghetti}         => {ground beef}       0.003066258  0.4600000 4.681764    23
## [5]  {frozen vegetables,
##       mineral water,
##       soup}              => {milk}              0.003066258  0.6052632 4.670863    23
## [6]  {chocolate,
##       herb & pepper}     => {ground beef}       0.003999467  0.4411765 4.490183    30
## [7]  {chocolate,
##       mineral water,
##       shrimp}            => {frozen vegetables} 0.003199573  0.4210526 4.417225    24
## [8]  {frozen vegetables,
##       mineral water,
##       olive oil}         => {milk}              0.003332889  0.5102041 3.937285    25
## [9]  {cereals,
##       ground beef}       => {spaghetti}         0.003066258  0.6764706 3.885303    23
## [10] {frozen vegetables,
##       soup}              => {milk}              0.003999467  0.5000000 3.858539    30
## [11] {chicken,
##       olive oil}         => {milk}              0.003599520  0.5000000 3.858539    27
## [12] {frozen smoothie,
##       mineral water,
##       spaghetti}         => {milk}              0.003199573  0.4705882 3.631566    24
## [13] {olive oil,
##       tomatoes}          => {spaghetti}         0.004399413  0.6111111 3.509912    33
## [14] {spaghetti,
##       whole wheat pasta} => {milk}              0.003999467  0.4545455 3.507763    30
## [15] {soup,
##       tomatoes}          => {milk}              0.003066258  0.4423077 3.413323    23
## [16] {chocolate,
##       frozen vegetables,
##       spaghetti}         => {milk}              0.003466205  0.4406780 3.400746    26
## [17] {ground beef,
##       tomato sauce}      => {spaghetti}         0.003066258  0.5750000 3.302508    23
## [18] {cooking oil,
##       ground beef}       => {spaghetti}         0.004799360  0.5714286 3.281995    36
## [19] {frozen vegetables,
##       olive oil}         => {milk}              0.004799360  0.4235294 3.268410    36
## [20] {ground beef,
##       mineral water,
##       tomatoes}          => {spaghetti}         0.003066258  0.5609756 3.221959    23
```

We can observe 281 rules with 40% confidence.

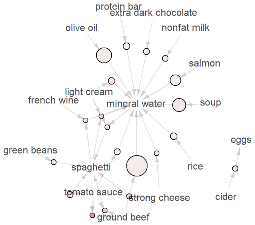
```
plot(rules[1:20],method = "graph",control = list(type = "items"))
```

```
## Warning: Unknown control parameters: type
```

```
## Available control parameters (with default values):
## main  =  Graph for 20 rules
## nodeColors    =  c("#66CC6680", "#9999CC80")
## nodeCol   =  c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF", "#EE0F0FFF", "#EE1212FF", "#EE
1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1EFF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF",
"#EE3131FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE4444FF", "#EE4747FF", "#EE4A4AF
F", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#EE5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6
666FF", "#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF",  "#EE7878FF", "#EE7B7BFF", "#EE7E7EFF",
"#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8BFF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999F
F", "#EE9B9BFF", "#EE9D9DFF", "#EE9F9FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EEA5A5FF", "#EEA7A7FF", "#EEA
9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#EEB0B0FF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF",
"#EEB8B8FF", "#EEBABAFF", "#EEBCBCFF", "#EEBDBDFF", "#EEBFBFFF", "#EEC1C1FF", "#EEC3C3FF", "#EEC4C4FF", "#EEC6C6F
F", "#EEC8C8FF",  "#EEC9C9FF", "#EECBCBFF", "#EECDCDFF", "#EECFCFFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EE
D5D5FF", "#EED7D7FF", "#EED9D9FF", "#EEDBDBFF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EEE1E1FF", "#EEE3E3FF",
"#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#EEECECFF", "#EEEEEEFF")
## edgeCol   =  c("#474747FF", "#494949FF", "#4B4B4BFF", "#4D4D4DFF", "#4F4F4FFF", "#515151FF", "#535353FF", "#55
5555FF", "#575757FF", "#595959FF", "#5B5B5BFF", "#5E5E5EFF", "#606060FF", "#626262FF", "#646464FF", "#666666FF",
"#686868FF", "#6A6A6AFF", "#6C6C6CFF", "#6E6E6EFF", "#707070FF", "#727272FF", "#747474FF", "#767676FF", "#787878F
F", "#7A7A7AFF", "#7C7C7CFF", "#7E7E7EFF", "#808080FF", "#828282FF", "#848484FF", "#868686FF", "#888888FF", "#8A8
A8AFF", "#8C8C8CFF", "#8D8D8DFF", "#8F8F8FFF", "#919191FF", "#939393FF",  "#959595FF", "#979797FF", "#999999FF",
"#9A9A9AFF", "#9C9C9CFF", "#9E9E9EFF", "#A0A0A0FF", "#A2A2A2FF", "#A3A3A3FF", "#A5A5A5FF", "#A7A7A7FF", "#A9A9A9F
F", "#AAAAAAFF", "#ACACACFF", "#AEAEAEFF", "#AFAFAFFF", "#B1B1B1FF", "#B3B3B3FF", "#B4B4B4FF", "#B6B6B6FF", "#B7B
7B7FF", "#B9B9B9FF", "#BBBBBBFF", "#BCBCBCFF", "#BEBEBEFF", "#BFBFBFFF", "#C1C1C1FF", "#C2C2C2FF", "#C3C3C4FF",
"#C5C5C5FF", "#C6C6C6FF", "#C8C8C8FF", "#C9C9C9FF", "#CACACAFF", "#CCCCCCFF", "#CDCDCDFF", "#CECECEFF", "#CFCFCFF
F", "#D1D1D1FF",  "#D2D2D2FF", "#D3D3D3FF", "#D4D4D4FF", "#D5D5D5FF", "#D6D6D6FF", "#D7D7D7FF", "#D8D8D8FF", "#D9
D9D9FF", "#DADADAFF", "#DBDBDBFF", "#DCDCDCFF", "#DDDDDDFF", "#DEDEDEFF", "#DEDEDEFF", "#DFDFDFFF", "#E0E0E0FF",
"#E0E0E0FF", "#E1E1E1FF", "#E1E1E1FF", "#E2E2E2FF", "#E2E2E2FF", "#E2E2E2FF")
## alpha     =  0.5
## cex   =  1
## itemLabels    =  TRUE
## labelCol  =  #000000B3
## measureLabels     =  FALSE
## precision     =  3
## layout    =  NULL
## layoutParams  =  list()
## arrowSize     =  0.5
## engine    =  igraph
## plot  =  TRUE
## plot_options  =  list()
## max   =  100
## verbose   =  FALSE
```

Graph for 20 rules

size: support (0.003 - 0.041)
color: lift (1.683 - 4.981)



The size of graph nodes is based on support levels and the colour on lift ratios. The incoming lines show the Antecedants or the LHS and the RHS is represented by names of items.

```
# COnsidering item to be bought 3 times a day that defines support as 0.003 and considering confidence 0.2 by def
ault value
#Support 3*7/7500 ~ 0.003
rules = apriori(data = dataset, parameter = list(support = 0.003, confidence = 0.2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.003      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 22
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [115 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [1348 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

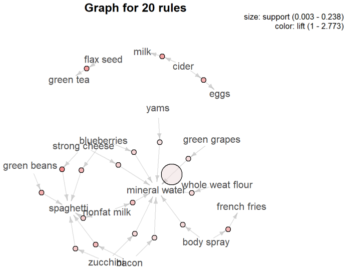
```
#Inspecitng top 20 rules with support 0.03 and confidence of 20%

inspect(sort(rules, by = 'lift')[1:20])
```

```
##      lhs                     rhs                      support confidence     lift count
## [1]  {mineral water,
##       whole wheat pasta} => {olive oil}         0.003866151  0.4027778 6.115863    29
## [2]  {frozen vegetables,
##       milk,
##       mineral water}     => {soup}              0.003066258  0.2771084 5.484407    23
## [3]  {fromage blanc}     => {honey}             0.003332889  0.2450980 5.164271    25
## [4]  {spaghetti,
##       tomato sauce}      => {ground beef}       0.003066258  0.4893617 4.980600    23
## [5]  {light cream}       => {chicken}           0.004532729  0.2905983 4.843951    34
## [6]  {pasta}             => {escalope}          0.005865885  0.3728814 4.700812    44
## [7]  {french fries,
##       herb & pepper}     => {ground beef}       0.003199573  0.4615385 4.697422    24
## [8]  {cereals,
##       spaghetti}         => {ground beef}       0.003066258  0.4600000 4.681764    23
## [9]  {frozen vegetables,
##       mineral water,
##       soup}              => {milk}              0.003066258  0.6052632 4.670863    23
## [10] {french fries,
##       ground beef}       => {herb & pepper}     0.003199573  0.2307692 4.665768    24
## [11] {chocolate,
##       frozen vegetables,
##       mineral water}     => {shrimp}            0.003199573  0.3287671 4.600900    24
## [12] {frozen vegetables,
##       milk,
##       mineral water}     => {olive oil}         0.003332889  0.3012048 4.573557    25
## [13] {pasta}             => {shrimp}            0.005065991  0.3220339 4.506672    38
## [14] {chocolate,
##       herb & pepper}     => {ground beef}       0.003999467  0.4411765 4.490183    30
## [15] {chocolate,
##       mineral water,
##       shrimp}            => {frozen vegetables} 0.003199573  0.4210526 4.417225    24
## [16] {cake,
##       frozen vegetables} => {tomatoes}          0.003066258  0.2987013 4.367560    23
## [17] {milk,
##       tomatoes}          => {soup}              0.003066258  0.2190476 4.335293    23
## [18] {eggs,
##       ground beef}       => {herb & pepper}     0.004132782  0.2066667 4.178455    31
## [19] {milk,
##       olive oil}         => {soup}              0.003599520  0.2109375 4.174781    27
## [20] {whole wheat pasta} => {olive oil}         0.007998933  0.2714932 4.122410    60
```

```
plot(rules[1:20],method = "graph",control = list(type = "items"))
```

```
## Warning: Unknown control parameters: type
```

```
## Available control parameters (with default values):
## main  =  Graph for 20 rules
## nodeColors    =  c("#66CC6680", "#9999CC80")
## nodeCol   =  c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF", "#EE0F0FFF", "#EE1212FF", "#EE
1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1EFF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF",
"#EE3131FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE4444FF", "#EE4747FF", "#EE4A4AF
F", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#EE5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6
666FF", "#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF",  "#EE7878FF", "#EE7B7BFF", "#EE7E7EFF",
"#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8BFF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999F
F", "#EE9B9BFF", "#EE9D9DFF", "#EE9F9FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EEA5A5FF", "#EEA7A7FF", "#EEA
9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#EEB0B0FF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF",
"#EEB8B8FF", "#EEBABAFF", "#EEBCBCFF", "#EEBDBDFF", "#EEBFBFFF", "#EEC1C1FF", "#EEC3C3FF", "#EEC4C4FF", "#EEC6C6F
F", "#EEC8C8FF",  "#EEC9C9FF", "#EECBCBFF", "#EECDCDFF", "#EECFCFFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EE
D5D5FF", "#EED7D7FF", "#EED9D9FF", "#EEDBDBFF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EEE1E1FF", "#EEE3E3FF",
"#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#EEECECFF", "#EEEEEEFF")
## edgeCol   =  c("#474747FF", "#494949FF", "#4B4B4BFF", "#4D4D4DFF", "#4F4F4FFF", "#515151FF", "#535353FF", "#55
5555FF", "#575757FF", "#595959FF", "#5B5B5BFF", "#5E5E5EFF", "#606060FF", "#626262FF", "#646464FF", "#666666FF",
"#686868FF", "#6A6A6AFF", "#6C6C6CFF", "#6E6E6EFF", "#707070FF", "#727272FF", "#747474FF", "#767676FF", "#787878F
F", "#7A7A7AFF", "#7C7C7CFF", "#7E7E7EFF", "#808080FF", "#828282FF", "#848484FF", "#868686FF", "#888888FF", "#8A8
A8AFF", "#8C8C8CFF", "#8D8D8DFF", "#8F8F8FFF", "#919191FF", "#939393FF",  "#959595FF", "#979797FF", "#999999FF",
"#9A9A9AFF", "#9C9C9CFF", "#9E9E9EFF", "#A0A0A0FF", "#A2A2A2FF", "#A3A3A3FF", "#A5A5A5FF", "#A7A7A7FF", "#A9A9A9F
F", "#AAAAAAFF", "#ACACACFF", "#AEAEAEFF", "#AFAFAFFF", "#B1B1B1FF", "#B3B3B3FF", "#B4B4B4FF", "#B6B6B6FF", "#B7B
7B7FF", "#B9B9B9FF", "#BBBBBBFF", "#BCBCBCFF", "#BEBEBEFF", "#BFBFBFFF", "#C1C1C1FF", "#C2C2C2FF", "#C3C3C4FF",
"#C5C5C5FF", "#C6C6C6FF", "#C8C8C8FF", "#C9C9C9FF", "#CACACAFF", "#CCCCCCFF", "#CDCDCDFF", "#CECECEFF", "#CFCFCFF
F", "#D1D1D1FF",  "#D2D2D2FF", "#D3D3D3FF", "#D4D4D4FF", "#D5D5D5FF", "#D6D6D6FF", "#D7D7D7FF", "#D8D8D8FF", "#D9
D9D9FF", "#DADADAFF", "#DBDBDBFF", "#DCDCDCFF", "#DDDDDDFF", "#DEDEDEFF", "#DEDEDEFF", "#DFDFDFFF", "#E0E0E0FF",
"#E0E0E0FF", "#E1E1E1FF", "#E1E1E1FF", "#E2E2E2FF", "#E2E2E2FF", "#E2E2E2FF")
## alpha     =  0.5
## cex   =  1
## itemLabels    =  TRUE
## labelCol  =  #000000B3
## measureLabels     =  FALSE
## precision     =  3
## layout    =  NULL
## layoutParams  =  list()
## arrowSize     =  0.5
## engine    =  igraph
## plot  =  TRUE
## plot_options  =  list()
## max   =  100
## verbose   =  FALSE
```



We can observe 1348 rules with 20% confidence.With this confidence we are getting better rules.

```
# COnsidering item to be bought 4 times a day that defines support as 0.004 and considering confidence 0.2 by def
ault value
#Support 4*7/7500 ~ 0.004
rules = apriori(data = dataset, parameter = list(support = 0.004, confidence = 0.2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   0.004      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 30
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [114 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [811 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

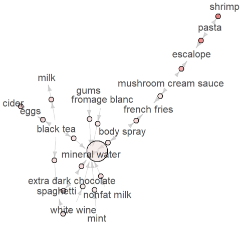
```
#Inspecitng top 20 rules with support 0.04 and confidence of 20%

inspect(sort(rules, by = 'lift')[1:20])
```

```
##      lhs                        rhs                   support confidence     lift count
## [1]  {light cream}           => {chicken}         0.004532729  0.2905983 4.843951    34
## [2]  {pasta}                 => {escalope}        0.005865885  0.3728814 4.700812    44
## [3]  {pasta}                 => {shrimp}          0.005065991  0.3220339 4.506672    38
## [4]  {eggs,
##       ground beef}           => {herb & pepper}   0.004132782  0.2066667 4.178455    31
## [5]  {whole wheat pasta}     => {olive oil}       0.007998933  0.2714932 4.122410    60
## [6]  {herb & pepper,
##       spaghetti}             => {ground beef}     0.006399147  0.3934426 4.004360    48
## [7]  {herb & pepper,
##       mineral water}         => {ground beef}     0.006665778  0.3906250 3.975683    50
## [8]  {tomato sauce}          => {ground beef}     0.005332622  0.3773585 3.840659    40
## [9]  {mushroom cream sauce} => {escalope}         0.005732569  0.3006993 3.790833    43
## [10] {frozen vegetables,
##       mineral water,
##       spaghetti}             => {ground beef}     0.004399413  0.3666667 3.731841    33
## [11] {olive oil,
##       tomatoes}              => {spaghetti}       0.004399413  0.6111111 3.509912    33
## [12] {frozen vegetables,
##       spaghetti}             => {tomatoes}        0.006665778  0.2392344 3.498046    50
## [13] {mineral water,
##       soup}                  => {olive oil}       0.005199307  0.2254335 3.423030    39
## [14] {ground beef,
##       milk}                  => {olive oil}       0.004932676  0.2242424 3.404944    37
## [15] {eggs,
##       herb & pepper}         => {ground beef}     0.004132782  0.3297872 3.356491    31
## [16] {spaghetti,
##       tomatoes}              => {frozen vegetables} 0.006665778  0.3184713 3.341054    50
## [17] {herb & pepper}         => {ground beef}     0.015997867  0.3234501 3.291994   120
## [18] {grated cheese,
##       spaghetti}             => {ground beef}     0.005332622  0.3225806 3.283144    40
## [19] {cooking oil,
##       ground beef}           => {spaghetti}       0.004799360  0.5714286 3.281995    36
## [20] {frozen vegetables,
##       olive oil}             => {milk}            0.004799360  0.4235294 3.268410    36
```

```
plot(rules[1:20],method = "graph",control = list(type = "items"))
```

```
## Warning: Unknown control parameters: type
```

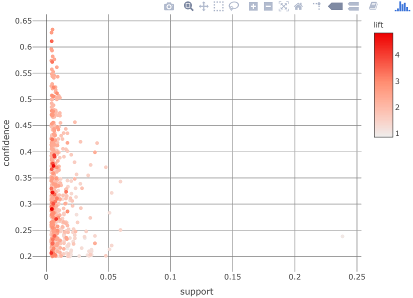
```
## Available control parameters (with default values):
## main  =  Graph for 20 rules
## nodeColors    =  c("#66CC6680", "#9999CC80")
## nodeCol   =  c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF", "#EE0F0FFF", "#EE1212FF", "#EE
1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1EFF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF",
"#EE3131FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE4444FF", "#EE4747FF", "#EE4A4AF
F", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#EE5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6
666FF", "#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF",  "#EE7878FF", "#EE7B7BFF", "#EE7E7EFF",
"#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8BFF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999F
F", "#EE9B9BFF", "#EE9D9DFF", "#EE9F9FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EEA5A5FF", "#EEA7A7FF", "#EEA
9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#EEB0B0FF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF",
"#EEB8B8FF", "#EEBABAFF", "#EEBCBCFF", "#EEBDBDFF", "#EEBFBFFF", "#EEC1C1FF", "#EEC3C3FF", "#EEC4C4FF", "#EEC6C6F
F", "#EEC8C8FF",  "#EEC9C9FF", "#EECBCBFF", "#EECDCDFF", "#EECFCFFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EE
D5D5FF", "#EED7D7FF", "#EED9D9FF", "#EEDBDBFF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EEE1E1FF", "#EEE3E3FF",
"#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#EEECECFF", "#EEEEEEFF")
## edgeCol   =  c("#474747FF", "#494949FF", "#4B4B4BFF", "#4D4D4DFF", "#4F4F4FFF", "#515151FF", "#535353FF", "#55
5555FF", "#575757FF", "#595959FF", "#5B5B5BFF", "#5E5E5EFF", "#606060FF", "#626262FF", "#646464FF", "#666666FF",
"#686868FF", "#6A6A6AFF", "#6C6C6CFF", "#6E6E6EFF", "#707070FF", "#727272FF", "#747474FF", "#767676FF", "#787878F
F", "#7A7A7AFF", "#7C7C7CFF", "#7E7E7EFF", "#808080FF", "#828282FF", "#848484FF", "#868686FF", "#888888FF", "#8A8
A8AFF", "#8C8C8CFF", "#8D8D8DFF", "#8F8F8FFF", "#919191FF", "#939393FF",  "#959595FF", "#979797FF", "#999999FF",
"#9A9A9AFF", "#9C9C9CFF", "#9E9E9EFF", "#A0A0A0FF", "#A2A2A2FF", "#A3A3A3FF", "#A5A5A5FF", "#A7A7A7FF", "#A9A9A9F
F", "#AAAAAAFF", "#ACACACFF", "#AEAEAEFF", "#AFAFAFFF", "#B1B1B1FF", "#B3B3B3FF", "#B4B4B4FF", "#B6B6B6FF", "#B7B
7B7FF", "#B9B9B9FF", "#BBBBBBFF", "#BCBCBCFF", "#BEBEBEFF", "#BFBFBFFF", "#C1C1C1FF", "#C2C2C2FF", "#C3C3C4FF",
"#C5C5C5FF", "#C6C6C6FF", "#C8C8C8FF", "#C9C9C9FF", "#CACACAFF", "#CCCCCCFF", "#CDCDCDFF", "#CECECEFF", "#CFCFCFF
F", "#D1D1D1FF",  "#D2D2D2FF", "#D3D3D3FF", "#D4D4D4FF", "#D5D5D5FF", "#D6D6D6FF", "#D7D7D7FF", "#D8D8D8FF", "#D9
D9D9FF", "#DADADAFF", "#DBDBDBFF", "#DCDCDCFF", "#DDDDDDFF", "#DEDEDEFF", "#DEDEDEFF", "#DFDFDFFF", "#E0E0E0FF",
"#E0E0E0FF", "#E1E1E1FF", "#E1E1E1FF", "#E2E2E2FF", "#E2E2E2FF", "#E2E2E2FF")
## alpha     =  0.5
## cex   =  1
## itemLabels    =  TRUE
## labelCol  =  #000000B3
## measureLabels     =  FALSE
## precision     =  3
## layout    =  NULL
## layoutParams  =  list()
## arrowSize     =  0.5
## engine    =  igraph
## plot  =  TRUE
## plot_options  =  list()
## max   =  100
## verbose   =  FALSE
```



```
#The plot uses the arulesViz package and plotly to generate an interactive plot. We can hover over each rule and
 see the Support, Confidence and Lift.

#As the interactive plot suggests, one rule that has a confidence of 0.61 is the one above. It has an exceptional
ly high lift as well, at 3.51.
plotly_arules(rules)
```

```
## Warning: 'plotly_arules' is deprecated.
## Use 'plot' instead.
## See help("Deprecated")
```

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



We can observe 811 rules with 20% confidence.With this confidence we are getting better and appropriate rules By visualising these rules and plots, we can come up with a more detailed explanation of how to make business decisions in retail environments. we can make some specific aisles now in my store to help customers pick products easily from one place and also boost the store sales simultaneously.

Person who purchased light cream has also purchased chicken 30% times. Person who purchased pasta has also purchased escalope and shrimp 37 and 32% times. Person who purchased herb & pepper has also purchased spaghetti 57% times. Person who purchased cooking oil,ground beef has also purchased ground beef 39% times.

This analysis would help us improve our store sales and make calculated business decisions for people both in a hurry and the ones leisurely shopping.