Market Basket Analysis

R Markdown

Loading the required libraries

```
library(arules)
library(arulesViz)
```

Let us begin. Loading the data set. Here we are not loading using read.csv() function. Why? If you take a look into the data set out data set does not have variables in it. So this might create problem when we load. The read.transactions() function changes the dataset into a sparse matrix. It makes each row represent a transaction and creates columns for each item that a customer might purchase. Electronidex sells 125 items, so the sparse matrix creates 125 columns. It also changes the data to binary. (1=item purchased in that transaction OR 0=no purchase.)

```
TransactionDataSet <-suppressMessages( read.transactions("ElectronidexTransactions2017.csv",format = c(

summary(TransactionDataSet)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  125 columns (items) and a density of 0.03506172
##
## most frequent items:
##                     iMac                HP Laptop CYBERPOWER Gamer Desktop
##                     2519                     1909                     1809
##            Apple Earpods        Apple MacBook Air                  (Other)
##                     1715                     1530                    33622
##
## element (itemset/transaction) length distribution:
## sizes
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##    2 2163 1647 1294 1021  856  646  540  439  353  247  171  119   77   72   56
##   16   17   18   19   20   21   22   23   25   26   27   29   30
##   41   26   20   10   10   10    5    3    1    1    3    1    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.000   3.000   4.383   6.000  30.000
##
## includes extended item information - examples:
##                             labels
## 1 1TB Portable External Hard Drive
## 2 2TB Portable External Hard Drive
## 3                   3-Button Mouse
```

To view data set as a whole use "inspect (DatasetName)". This is gonna take a lot of time.

```
inspect (TransactionDataSet[1:5])
```

```
##     items
```

```
## [1] {Acer Aspire,
##      Belkin Mouse Pad,
##      Brother Printer Toner,
##      VGA Monitor Cable}
## [2] {Apple Wireless Keyboard,
##      Dell Desktop,
##      Lenovo Desktop Computer}
## [3] {iMac}
## [4] {Acer Desktop,
##      Intel Desktop,
##      Lenovo Desktop Computer,
##      XIBERIA Gaming Headset}
## [5] {ASUS Desktop,
##      Epson Black Ink,
##      HP Laptop,
##      iMac}
```

```
LIST(TransactionDataSet[1:5])#Lists the transactions by conversion (LIST must be capitalized)
```

```
## [[1]]
## [1] "Acer Aspire"           "Belkin Mouse Pad"      "Brother Printer Toner"
## [4] "VGA Monitor Cable"
##
## [[2]]
## [1] "Apple Wireless Keyboard" "Dell Desktop"
## [3] "Lenovo Desktop Computer"
##
## [[3]]
## [1] "iMac"
##
## [[4]]
## [1] "Acer Desktop"            "Intel Desktop"
## [3] "Lenovo Desktop Computer" "XIBERIA Gaming Headset"
##
## [[5]]
## [1] "ASUS Desktop"    "Epson Black Ink" "HP Laptop"       "iMac"
```

```
length(TransactionDataSet) # length of transaction
```

```
## [1] 9835
```

```
size(TransactionDataSet[1:10]) #No:of items per transaction upto the 10th row
```

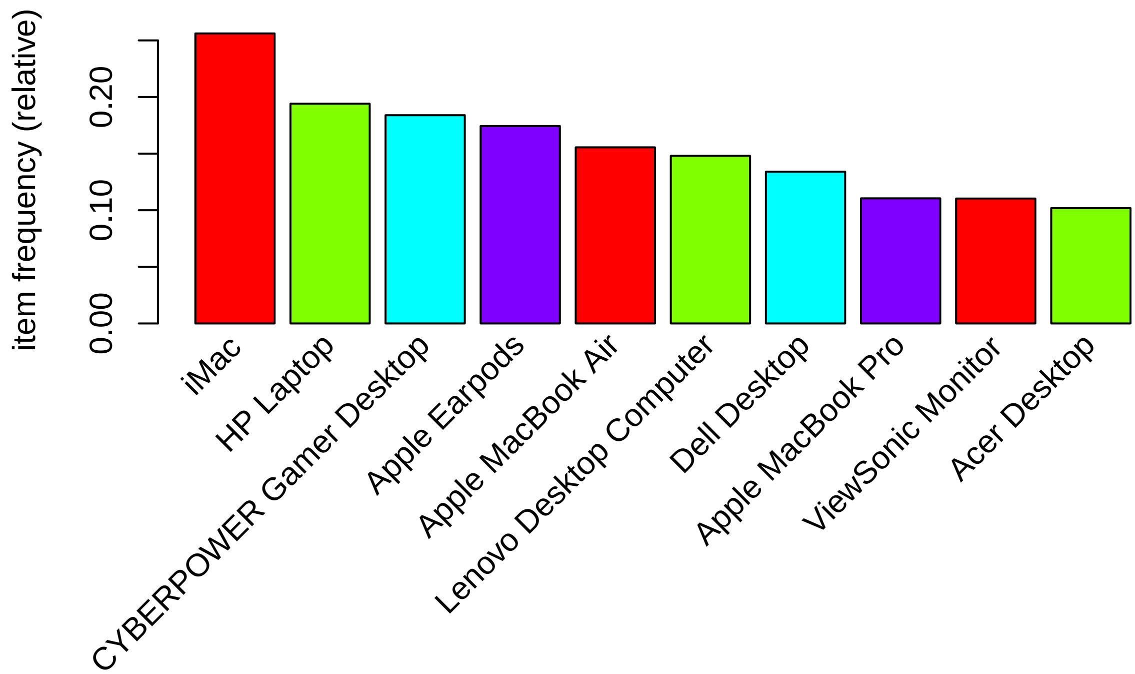
```
##  [1] 4 3 1 4 4 5 1 5 1 2
```

```
length(itemLabels(TransactionDataSet))# To see the length of item labels.. or lets say the list od labe
```

```
## [1] 125
```

There is total of 125 items & our data set has 9835 transactions

```
itemFrequencyPlot(TransactionDataSet, topN =10, col = rainbow(4),type = "relative")
```

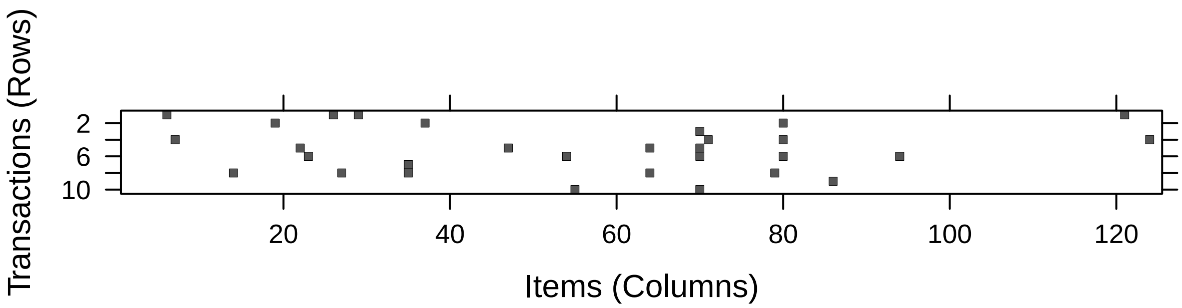


Let us find the list of items which are less popular.The below shows the list of 6 items with less sales volume:

```
##         Logitech Wireless Keyboard                  VGA Monitor Cable
##                        0.002236909                        0.002236909
## Panasonic On-Ear Stereo Headphones   1TB Portable External Hard Drive
##                        0.002338587                        0.002745297
##                          Canon Ink            Logitech Stereo Headset
##                        0.002745297                        0.003050330
```

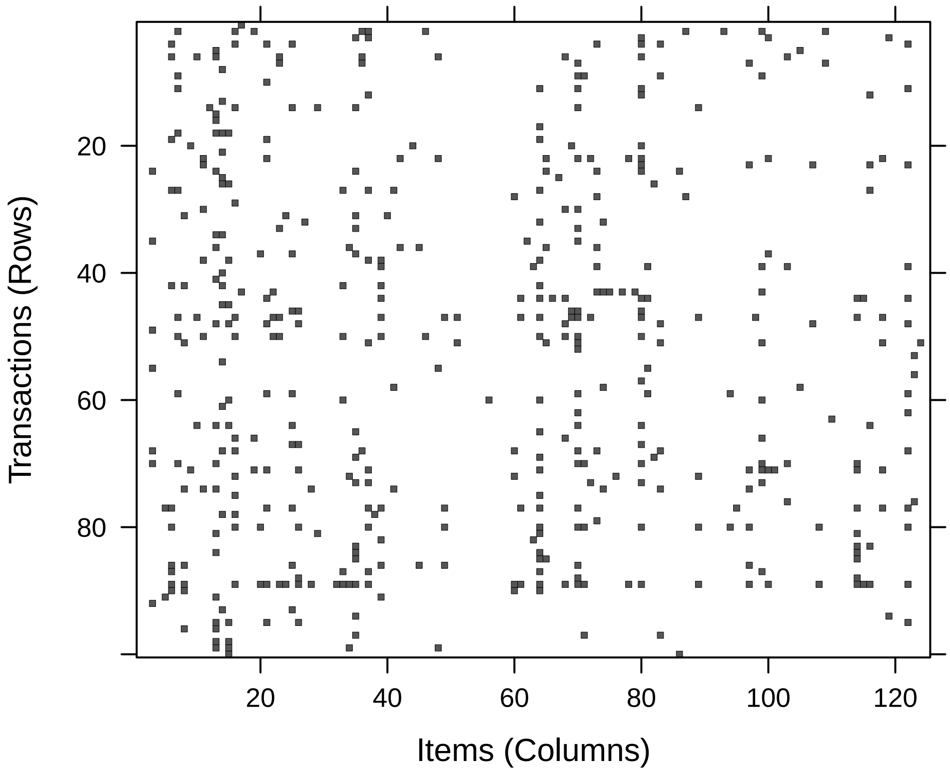
In addition to looking at the items, it's also possible to visualize the entire sparse matrix. To do so, use the image() function. The command to display the sparse matrix for the first 10 transactions is as follows:

```
image(TransactionDataSet[1:10],
      xlab = "Items (Columns)",
    ylab = "Transactions (Rows)")
```



this visualization will not be as useful for extremely large transaction databases, because the cells will be too small to discern. Still, by combining it with the sample() function, you can view the sparse matrix for a randomly sampled set of transactions. The command to create random selection of 100 transactions is as follows:

```
image(sample(TransactionDataSet,100))
```



From the graph we can understand that there is some popular items in the store as few columns seem fairly heavily populated.But overall, the distribution of dots seems fairly random.

Apiorifunction

```
rules1 <- apriori(TransactionDataSet,parameter = list(supp = 0.005, conf =0.6,minlen = 1,maxlen=10,targe
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.005      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: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[125 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 done [0.01s].
```

```
## writing ... [28 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
rules1
```

```
## set of 28 rules
```

```r
rules1 <- sort(rules1, by = 'lift')
```

These parameters are requesting that the rules cover 10% of the transactions and are 80% correct.

To View the rule

```r
inspect(rules1[1:5])
```

```
##     lhs                             rhs             support confidence    coverage     lift count
## [1] {Acer Aspire,
##      Dell Desktop,
##      ViewSonic Monitor}       => {HP Laptop} 0.005287239  0.8125000 0.006507372 4.185928    52
## [2] {Acer Aspire,
##      iMac,
##      ViewSonic Monitor}       => {HP Laptop} 0.006202339  0.6630435 0.009354347 3.415942    61
## [3] {Acer Desktop,
##      iMac,
##      ViewSonic Monitor}       => {HP Laptop} 0.006405694  0.6363636 0.010066090 3.278489    63
## [4] {Dell Desktop,
##      Lenovo Desktop Computer,
##      ViewSonic Monitor}       => {HP Laptop} 0.006202339  0.6224490 0.009964413 3.206802    61
## [5] {Computer Game,
##      ViewSonic Monitor}       => {HP Laptop} 0.007422471  0.6186441 0.011997966 3.187200    73
```

```r
#str(rules_df)
```

Receiving 0 rules means that you will need to experiment with the Support and Confidence values.

Now we recieved : set of 28 rules

When you're experimenting keep in mind:

1. If these values are too high, you will receive no rules or non-helpful rules.
2. If these values are too low, your computational time/memory will suffer, or you'll receive too many rules.
3. To get 'strong' rules, increase the value of 'conf' parameter.

Evalauting & taking a deep look

```r
summary(rules1)
```

```
## set of 28 rules
##
## rule length distribution (lhs + rhs):sizes
##  3  4
## 17 11
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.000   3.000   3.393   4.000   4.000
##
## summary of quality measures:
##     support            confidence        coverage              lift
##  Min.   :0.005084   Min.   :0.6000   Min.   :0.006507   Min.   :2.343
```

```
##  1st Qu.:0.005465   1st Qu.:0.6124   1st Qu.:0.008948   1st Qu.:2.423
##  Median :0.006355   Median :0.6321   Median :0.009964   Median :2.536
##  Mean   :0.006758   Mean   :0.6460   Mean   :0.010582   Mean   :2.725
##  3rd Qu.:0.007550   3rd Qu.:0.6648   3rd Qu.:0.012125   3rd Qu.:2.940
##  Max.   :0.010778   Max.   :0.8125   Max.   :0.017895   Max.   :4.186
##      count
##  Min.   : 50.00
##  1st Qu.: 53.75
##  Median : 62.50
##  Mean   : 66.46
##  3rd Qu.: 74.25
##  Max.   :106.00
##
## mining info:
##                data ntransactions support confidence
##  TransactionDataSet          9835   0.005        0.6
```

The summary of the rules gives us some very interesting information: 1. The number of rules: 28. 2. The distribution of rules by length: a length of 3 items has the most rules. 3. The summary of quality measures: ranges of support, confidence, and lift. 4. The information on data mining: total data mined, and the minimum parameters we set earlier.

Removing the redundant

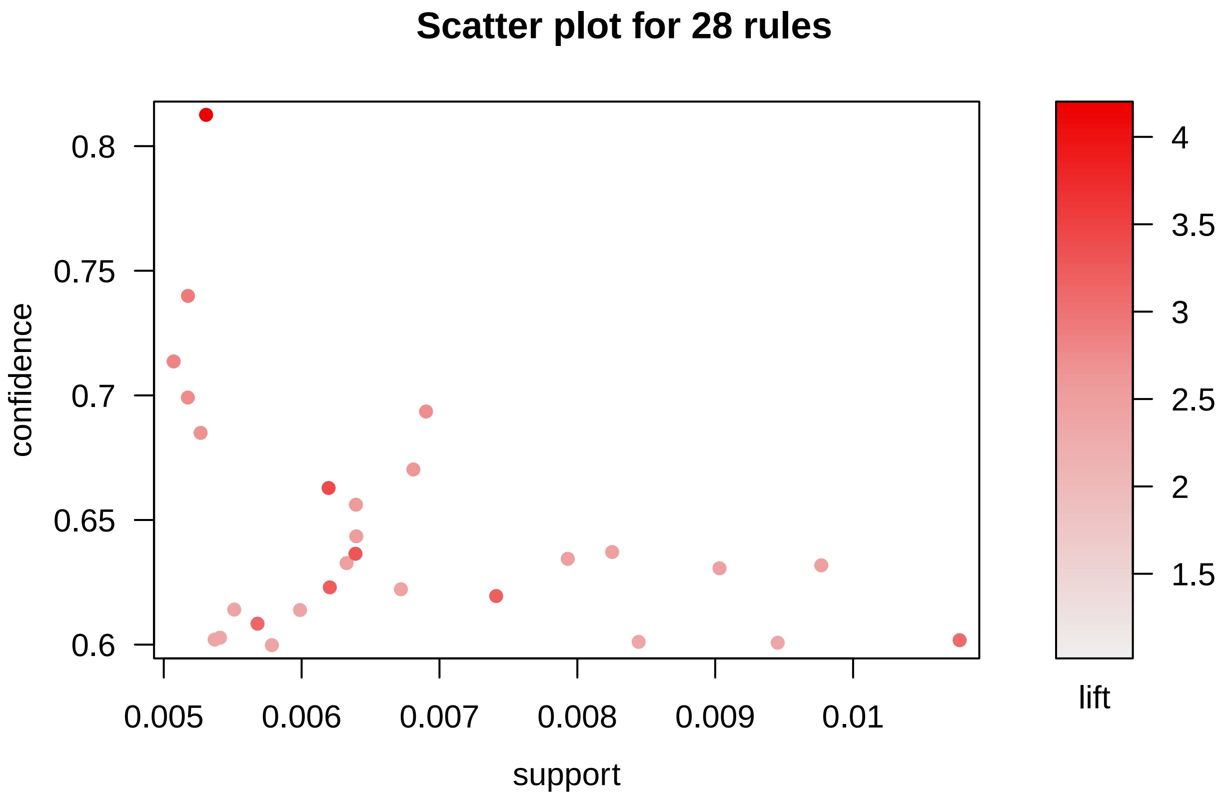
```r
table(is.redundant(rules1))
```

```
##
## FALSE
##    28
```

Ploting the 10 rules

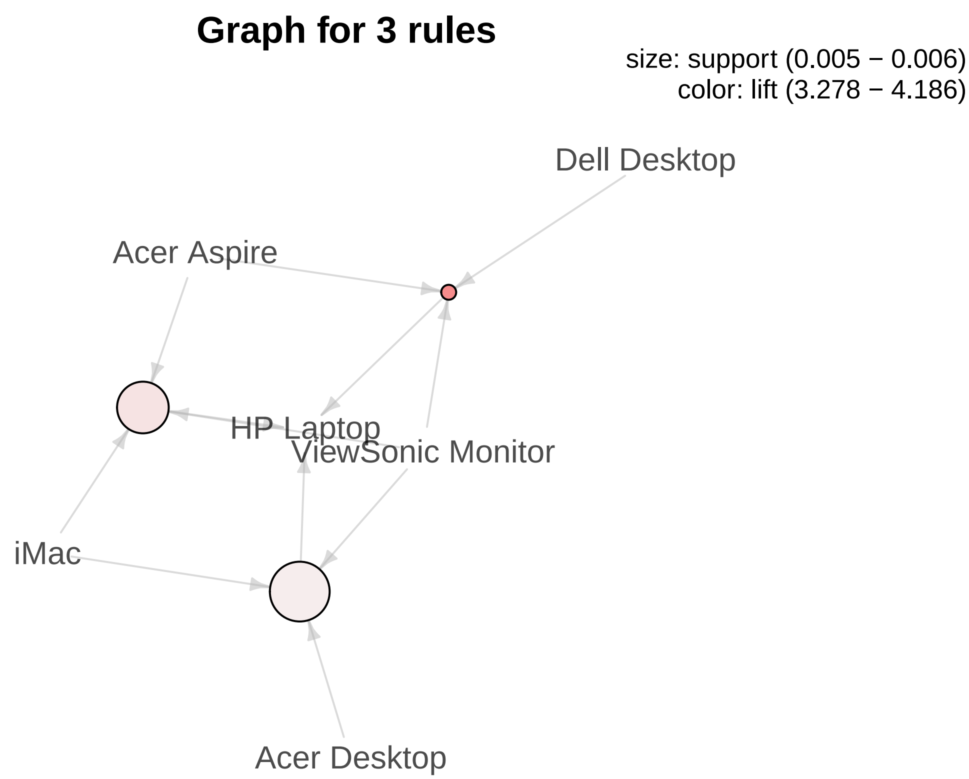
```r
topRules <- rules1[1:5]
plot(rules1)
```

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

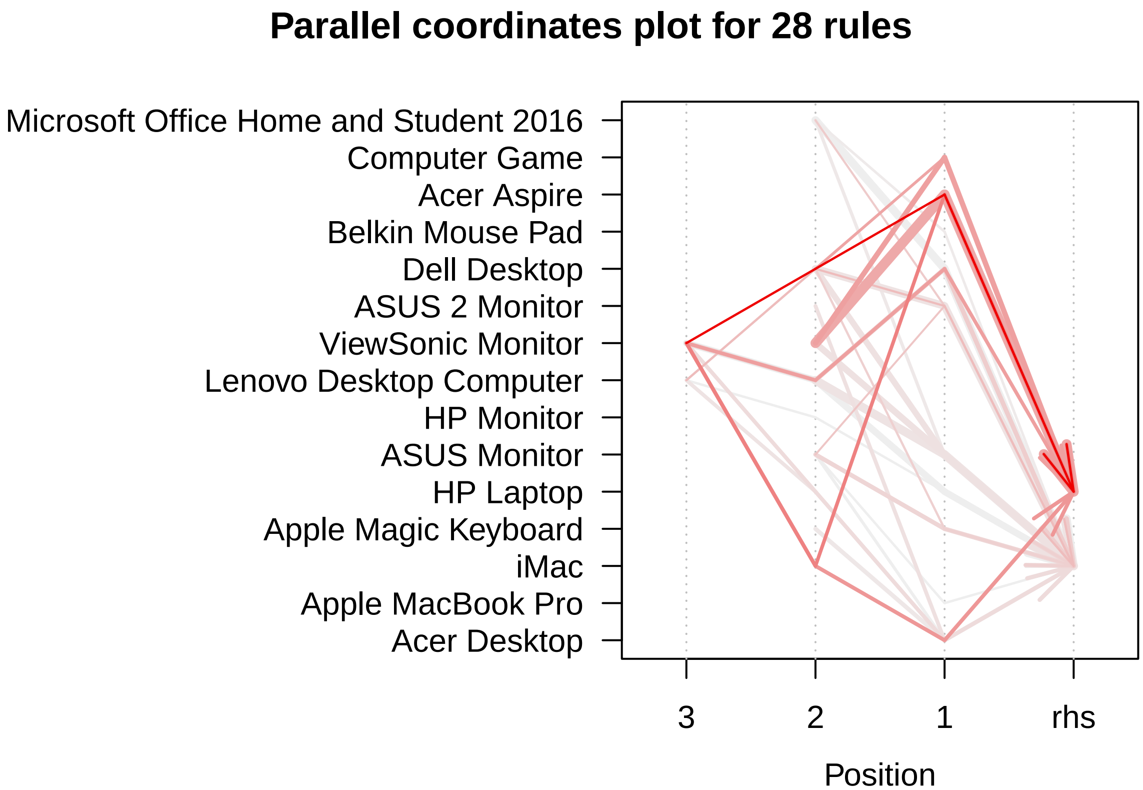


plot(rules1[1:3], method = “graph”, control = list(type= “items”))

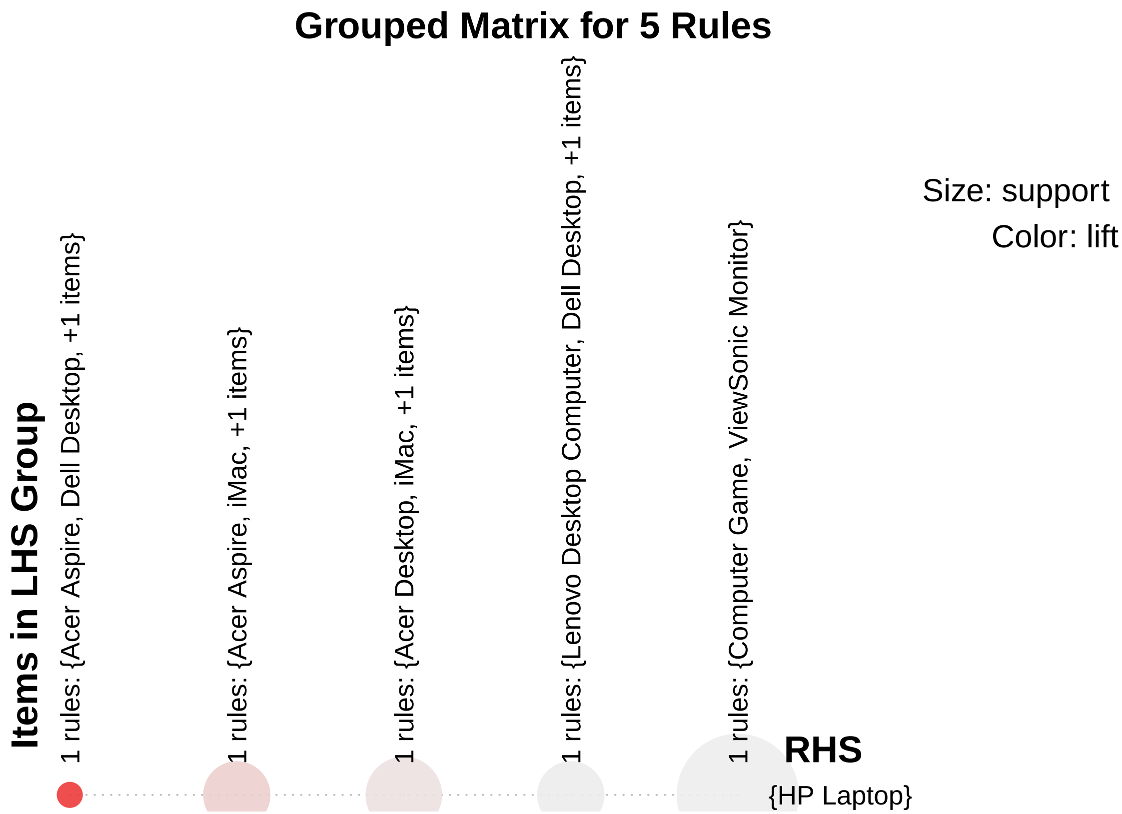
```
## Warning: Unknown control parameters: type
```




```
plot(rules1, method="paracoord", control=list(reorder=TRUE))
```




```
plot(topRules, method = "grouped")
```




```
#plot(rules1,method="graph",engine='interactive' ,shading=NA)
```

The interactive mode performs better but can not be displayed in knit mode.

```
ItemRules <- subset(rules1, items %in% "HP Laptop")
inspect(ItemRules[1:5])
```

```
##     lhs                          rhs             support confidence    coverage     lift count
## [1] {Acer Aspire,
##      Dell Desktop,
##      ViewSonic Monitor}       => {HP Laptop} 0.005287239  0.8125000 0.006507372 4.185928    52
## [2] {Acer Aspire,
##      iMac,
##      ViewSonic Monitor}       => {HP Laptop} 0.006202339  0.6630435 0.009354347 3.415942    61
## [3] {Acer Desktop,
##      iMac,
##      ViewSonic Monitor}       => {HP Laptop} 0.006405694  0.6363636 0.010066090 3.278489    63
## [4] {Dell Desktop,
##      Lenovo Desktop Computer,
##      ViewSonic Monitor}       => {HP Laptop} 0.006202339  0.6224490 0.009964413 3.206802    61
## [5] {Computer Game,
##      ViewSonic Monitor}       => {HP Laptop} 0.007422471  0.6186441 0.011997966 3.187200    73
```

```
rules_df <- as(rules1, "data.frame")
str(rules_df)
```

```
## 'data.frame':    28 obs. of  6 variables:
##  $ rules     : chr  "{Acer Aspire,Dell Desktop,ViewSonic Monitor} => {HP Laptop}" "{Acer Aspire,iMac
```

```
##  $ support   : num  0.00529 0.0062 0.00641 0.0062 0.00742 ...
##  $ confidence: num  0.812 0.663 0.636 0.622 0.619 ...
##  $ coverage  : num  0.00651 0.00935 0.01007 0.00996 0.012 ...
##  $ lift      : num  4.19 3.42 3.28 3.21 3.19 ...
##  $ count     : int  52 61 63 61 73 56 106 51 50 51 ...
```

```
write(rules1, file = "rules_df.csv", sep = ",", quote = TRUE, row.names = FALSE)
```

Now we know Laptop & Imax… lets understand if we can use them to purchase lower frequency products

```
rules_highest_lhs <- apriori(data= TransactionDataSet, parameter = list(supp = 0.01, conf=0.2),appearan
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.01      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: 98
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[125 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [82 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [4 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

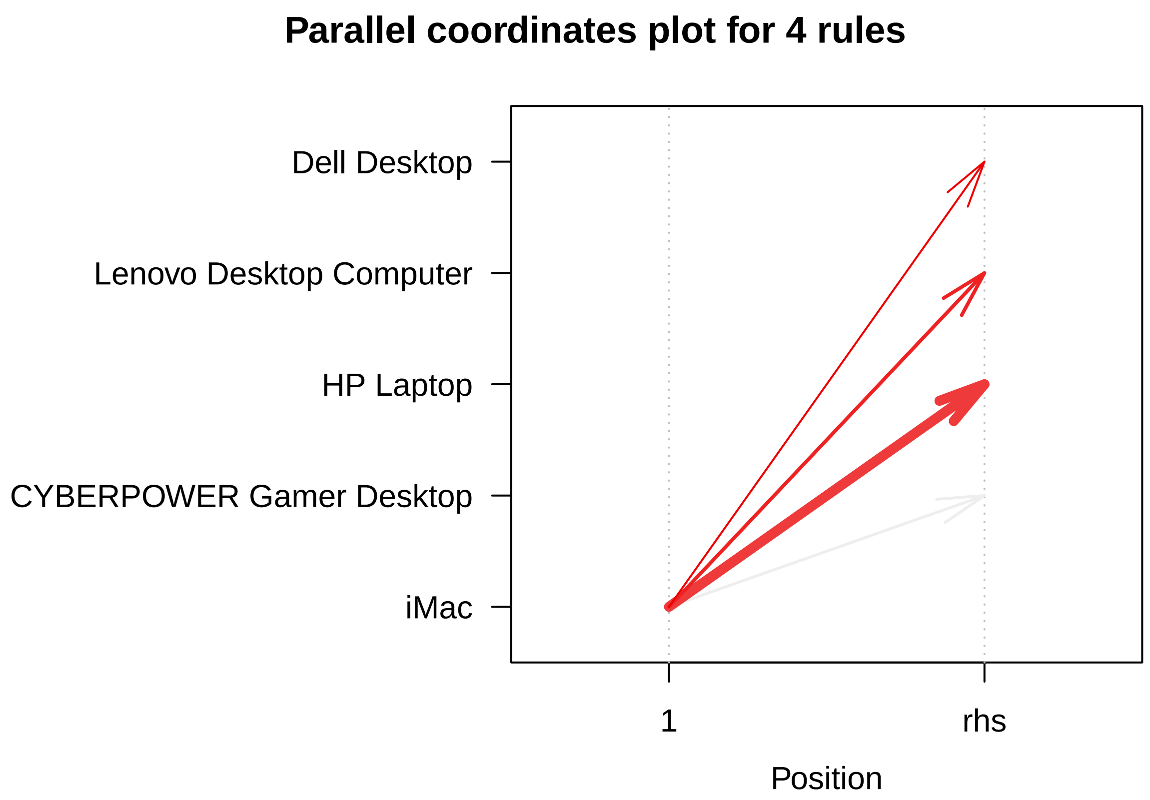
```
rules_highest_lhs
```

```
## set of 4 rules
```

```
rules_highest_lhs <- sort(rules_highest_lhs, by ="lift")
inspect(rules_highest_lhs)
```

```
##     lhs       rhs                          support    confidence coverage
## [1] {iMac} => {Dell Desktop}               0.05460092 0.2131798  0.2561261
## [2] {iMac} => {Lenovo Desktop Computer}    0.05876970 0.2294561  0.2561261
## [3] {iMac} => {HP Laptop}                  0.07554652 0.2949583  0.2561261
## [4] {iMac} => {CYBERPOWER Gamer Desktop} 0.05673615 0.2215165  0.2561261
##     lift     count
## [1] 1.590762 537
## [2] 1.549932 578
## [3] 1.519599 743
## [4] 1.204320 558
```

```
plot(rules_highest_lhs, method="paracoord", control=list(reorder=TRUE))
```




```
rules_highest_lhs <- apriori(data= TransactionDataSet, parameter = list(supp = 0.03, conf=0.2),appearanc
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.03      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: 295
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[125 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [43 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [6 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
rules_highest_lhs
```

```
## set of 6 rules
```

```r
rules_highest_lhs <- sort(rules_highest_lhs, by ="lift")
inspect(rules_highest_lhs)
```

```
##     lhs             rhs                          support    confidence coverage
## [1] {HP Laptop} => {ViewSonic Monitor}        0.04799187 0.2472499  0.1941027
## [2] {HP Laptop} => {Dell Desktop}             0.04494154 0.2315348  0.1941027
## [3] {HP Laptop} => {Lenovo Desktop Computer}  0.04616167 0.2378208  0.1941027
## [4] {HP Laptop} => {iMac}                     0.07554652 0.3892090  0.1941027
## [5] {HP Laptop} => {CYBERPOWER Gamer Desktop} 0.04260295 0.2194866  0.1941027
## [6] {}          => {iMac}                     0.25612608 0.2561261  1.0000000
##     lift     count
## [1] 2.241200  472
## [2] 1.727728  442
## [3] 1.606434  454
## [4] 1.519599  743
## [5] 1.193284  419
## [6] 1.000000 2519
```

```r
plot(rules_highest_lhs, method="paracoord", control=list(reorder=TRUE))
```