Week 4

R Packages

The packages you will need to install for the week are **Matrix**, **arules** and **arulesViz**.

Strawberry Pop-tarts During Hurricanes

If you read *Big Data* by Schonberger-Mayer and Cukier, then you are familiar with the Walmart story about stocking stores with strawberry pop-tarts during hurricanes. If not, here is the original story. We would expect that people buy sandbags, flashlights, and other similar items to prepare for hurricanes. Walmart figured out that people buy strawberry pop-tarts too!

Let's not forget the Target story about predicting pregnant customers. Target arrived at their answers by examining what known pregnant customers were buying throughout their pregnancies. The company then extrapolated the information onto other customers with unconfirmed pregnancy status.

Our task for this evening is in the similar vein. We will be looking at datasets to see if we can find regularities in them.

Frequent Pattern Analysis (Association Rule Mining)

We introduced unsupervised learning last week via cluster analysis. In particular, we examined k-means, k-medoids, and hierarchical clustering. We will examine another unsupervised learning method this week called **frequent pattern analysis (FPA)**.

Applications of FPA include analyses of market basket, DNA sequence, click stream analysis, and marketing activities (sale campaign; cross-marketing; etc.). We are going to examine two FPA algorithms: **Apriori** and **ECLAT**. **Apriori** is the more popular algorithm, but it can take up a lot of computational resources. **ECLAT** is a more efficient algorithm (i.e. faster) on smaller datasets.

Learning Goal for the Week

What interesting rules can we discover from 9,835 transactions containing 169 grocery products?

The Dataset

The *Groceries* dataset contains 9,835 transactions of 169 aggregated categories at a grocery store. The data was collected over a one month period.

Source of dataset:

Michael Hahsler, Kurt Hornik, and Thomas Reutterer (2006) Implications of probabilistic data modeling for mining association rules. In M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nuernberger, and W. Gaul, editors, From Data and Information Analysis to Knowledge Engineering, Studies in Classification, Data Analysis, and Knowledge Organization, pages 598-605. Springer-Verlag.

Basic Concepts

Assume we have the following transaction database.

An **itemset** is a list containing one or more items from the dataset.

Here are all the possible 1-itemset from the database above:

1-itemset: {beer},{nuts},{diaper},{coffee},{eggs},{milk}

Here are all the possible 2-itemsets from the database above:

2-itemset: {beer, nuts}; {beer, diaper}; {beer, coffee}; {beer, eggs}; {beer, milk}; {nuts, diaper}; {nuts, coffee}; {nuts, eggs}; {nuts, milk}; {diaper, coffee}; {diapers, eggs}; {diapers, milk}; {coffee, eggs}; {coffee, milk}; {eggs, milk}

3-itemset and 4-itemset are created in similar fashions.

Absolute Support is the count of occurrences of itemset X. For example, the absolute support for the 2-itemset {beer, diaper} is 3. The absolute support for the 2-itemset {eggs, milk} is 2.

Relative Support is the fraction of transactions that contain itemset X. For example, the relative support for the 2-itemset {beer, diaper} is 0.6

```
Relative Support = Count(X)/N = 3/5 = 0.6
```

An itemset is said to be **frequent** if its support >= minimum support threshold (minsup). The minsup value is set by the user and should reflect business knowledge.

FPA results in a set of association rules. A typical association rule would state that given itemset X, then itemset Y is likely to occur. For example, {diaper} -> {beer}. Customers who purchase diapers are likely to purchase beer.

Association rules are determined based on two quality measures: **support** and **confidence**.

```
Support: How often does the rule happen?
```

Agrawal, Imielinski, and Swami (1993) noted that support is equivalent to the concept of "statistical significance." The calculation is already discussed above.

```
Confidence: How often is the rule correct?
```

Agrawal, Imielinski, and Swami (1993) said confidence is the "rule's strength."

Confidence is calculated as follows:

```
confidence(X->Y)=(support(X,Y))/(support(X))
```

The user sets the **minimum support (minsup)** and **minimum confidence (minconf)** thresholds. **Rule interestingness** is determined by the minsup and minconf. Applied algorithms only report association rules that meet or exceed the minsup and minconf thresholds.

For example, let's say we set the minsup = 50% and minconf=50%

Here are two association rules that are "interesting."

```
{diaper}->{beer}

support(diaper,beer)=(count(diaper,beer))/N=3/5=0.6

confidence(diaper,beer)=(support(diaper,beer))/(support(diaper))=3/4=0.75

We report this rule as follows: diapers->beer (60%, 75%)
```

```
{beer}->{diaper}

support(beer,diaper)=(count(beer,diaper))/N=3/5=0.6

confidence(beer,diaper)=support(beer,diaper)/support(3) =3/3=1.0

We report this rule as follows: beer -> diapers (60%, 100%)
```

How Do FPA Algorithms Work?

FPA algorithms utilize a search tree to generate frequent itemsets. A search tree starts with an empty itemset in its initialization. Using the minsup threshold established by the user, a set of candidate itemsets are generated. Support for each candidate itemset is then generated. If a search tree tries to generate all possible candidate itemsets, the process would be very computationally intensive for large datasets. As a result, most FPA algorithms utilize the **downward closure property**. Downward closure states that a supersede itemset cannot be frequent if its subset itemsets are not frequent. Consequently, most FPA algorithms will only generate candidates and count support for those itemsets that meet the downward closure property.

Below is an illustration of the search tree that applies the downward closure property. We assume here that the minsup = 50%. Notice that no 3-itemset candidates are generated because only one 2-itemset {beer, diapers} is frequent.

Getting Started

The original data frame has 9,835 transactions with 169 grocery products. This translates into a sparse matrix with 9,835 rows and 169 columns.

```r
library(Matrix)
library(arules)
```

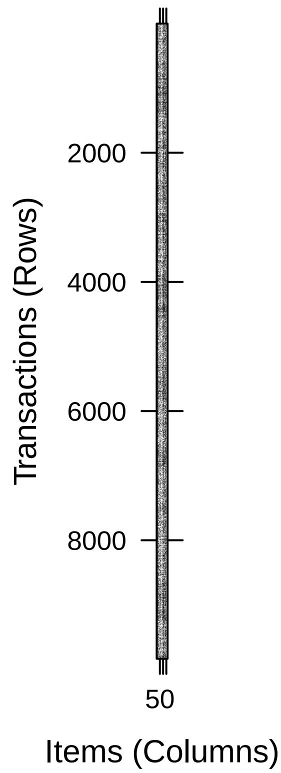
```
##
## Attaching package: 'arules'

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

```r
setwd("C:/Users/corylowe/OneDrive/Code/R Practice Code/Applied Data Mining_Portfolio/Week 4")
groceries <- read.transactions("groceries.csv", sep = ",") #9,835 transactions with 169 products.
```

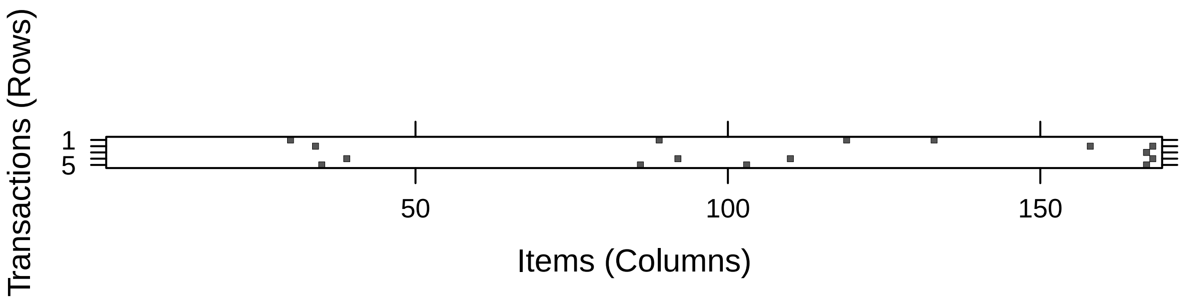
Let's visualize the sparse matrix. You cannot make much sense of this picture.

```r
image(groceries)
```



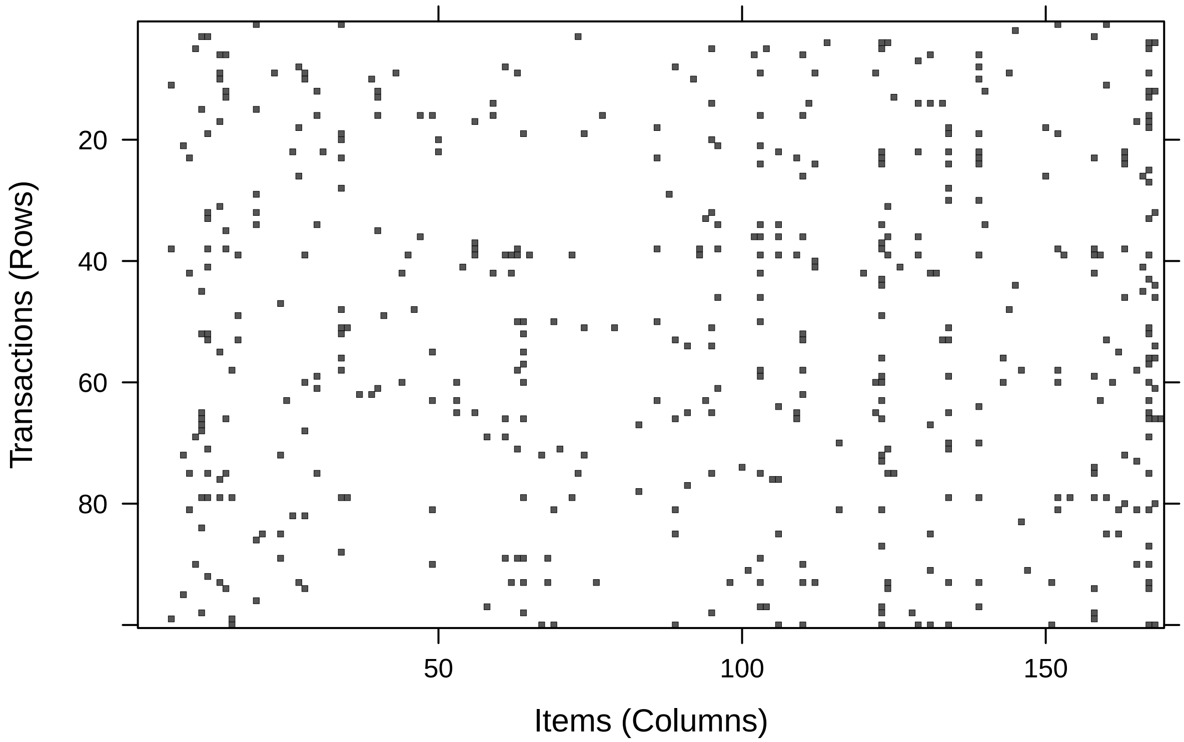
Let's narrow it down to the first five transactions.

```r
image(groceries[1:5])
```



Another visualization. This time of a random sample of 100 transactions.

```
image(sample(groceries, 100))
```



Can we quickly summarize the sparse matrix? Yes!

```r
summary(groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55
##   16   17   18   19   20   21   22   23   24   26   27   28   29   32
##   46   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##             labels
```

```
## 1 abrasive cleaner
## 2 artif. sweetener
## 3   baby cosmetics
```

Density indicates that only 2% of the elements are non-zero.

Mean is the average items per transaction.

Sum of "most frequent items" = total number of items bought in grocery dataset

Element is the frequency of a given number of items (i.e. 1, 2, 3, 4, etc.) bought in the transactions.

Here is another useful function that allows you to examine individual transactions.

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

```
##     items
## [1] {citrus fruit,
##      margarine,
##      ready soups,
##      semi-finished bread}
## [2] {coffee,
##      tropical fruit,
##      yogurt}
## [3] {whole milk}
## [4] {cream cheese,
##      meat spreads,
##      pip fruit,
##      yogurt}
## [5] {condensed milk,
##      long life bakery product,
##      other vegetables,
##      whole milk}
```

Examining 1-itemset

Let's count the **support** (or frequency) of the grocery items. We will put the support in a data frame so we can view them.

```
freq_groceries_data_frame <- as.data.frame(itemFrequency(groceries))
#View(freq_groceries_data_frame)
```

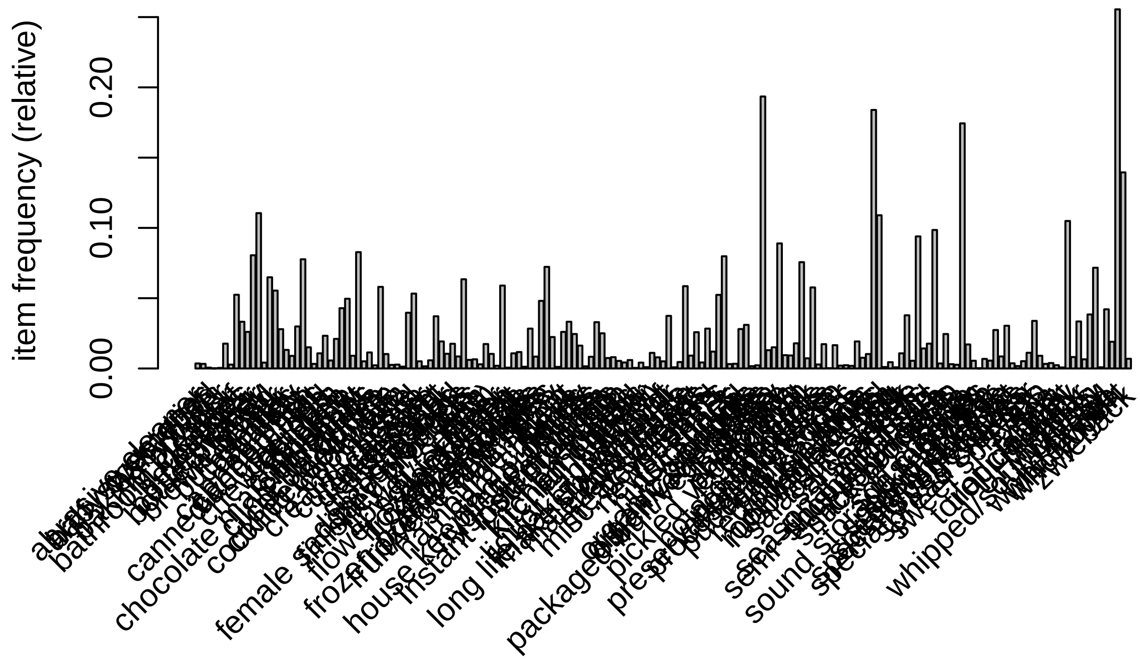
Let's pare this list down a bit to look at the first 15 items.

```
itemFrequency(groceries[, 1:15])
```

```
## abrasive cleaner artif. sweetener   baby cosmetics        baby food
##     0.0035587189     0.0032536858     0.0006100661     0.0001016777
##             bags    baking powder bathroom cleaner             beef
##     0.0004067107     0.0176919166     0.0027452974     0.0524656838
##          berries        beverages     bottled beer    bottled water
##     0.0332486019     0.0260294865     0.0805287239     0.1105236401
##           brandy      brown bread           butter
##     0.0041687850     0.0648703610     0.0554143366
```

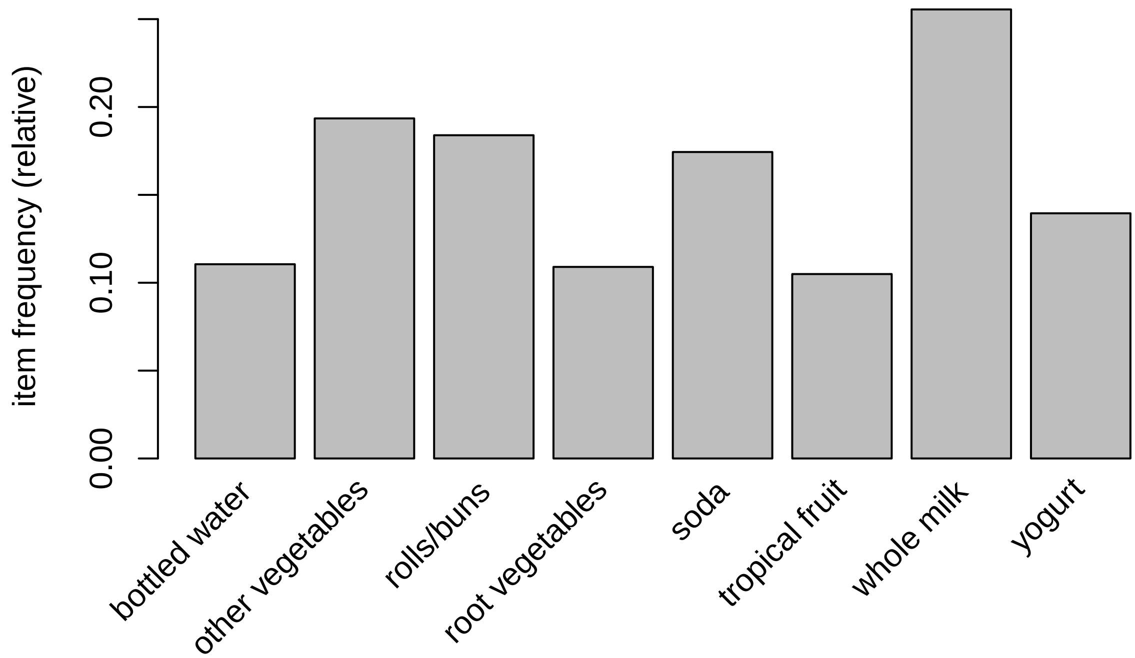
Plotting the support.

```r
itemFrequencyPlot(groceries)
```



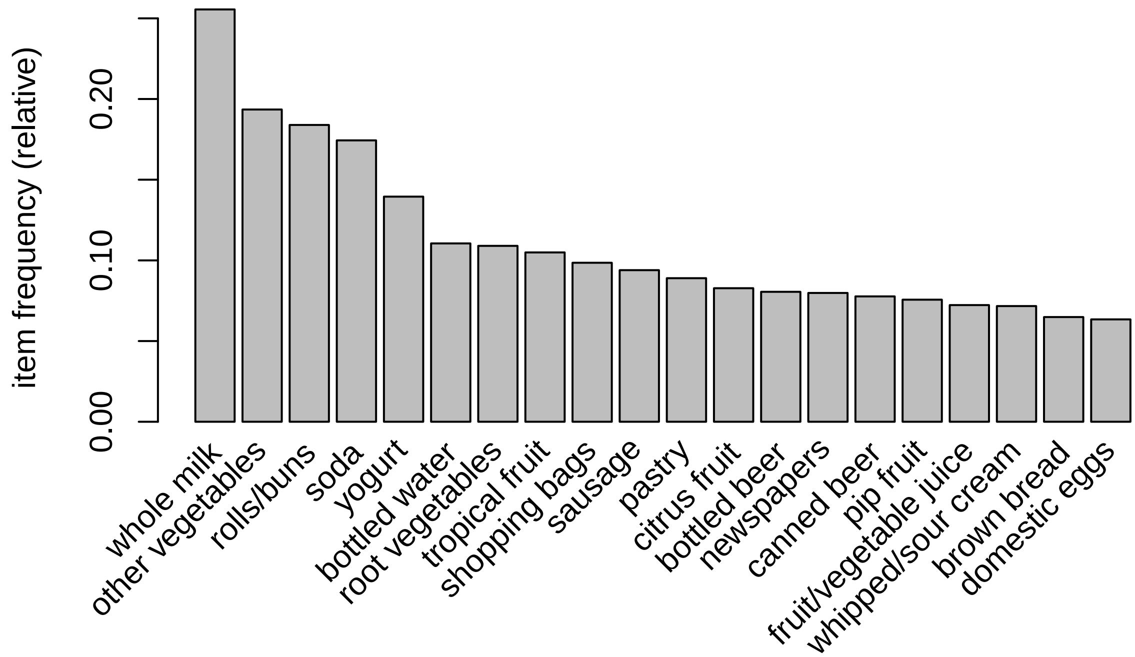
Too much information! Let's impose a rule. Minsup = 10%

```r
itemFrequencyPlot(groceries, support = 0.1)
```



Here is a different take. Let's say we want to look at the "top 20" items.

```
itemFrequencyPlot(groceries, topN = 20)
```



Apriori Algorithm

The most frequently used FPA algorithm is Apriori.

Pro: scalable for large datasets.

Con: computationally intensive. We have to keep comparing the candidate itemsets against the database until no frequent and/or candidate itemsets can be generated.

Data format requirement: Horizontal. One column has the tidset number (tid= transaction ID). Another column has a list of items.

Method

1. Initialize by scanning the database once to get frequent 1-itemset
2. Generate length (k+1) candidate itemsets from length k frequent itemsets
3. Test the candidate itemsets against the database. Prune candidate itemsets based on the minimum support threshold (minsup).
4. Terminate when no frequent or candidate set can be generated.

Exploring apriori() in arules Package

```
#?apriori
```

Default parameter settings

support = 0.1 (or 10%) confidence = 0.8 (or 80%) maxlen = maximum number of items in a rule. Default is 10. minlen = minimum number of items in a rule. Default is 1.

Let's try the default parameter settings first.

```
apriori(groceries)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      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: 983
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
## set of 0 rules
```

Not a single rule found! Let's try again with some tweakings to the parameter settings.

```
groceryrules <- apriori(groceries, parameter = list(support =
                          0.001, confidence = 0.5, minlen = 2))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5   0.001      2
##  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: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
```

```
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [5668 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Let's count the number of rules found

```r
print(groceryrules)
```

```
## set of 5668 rules
```

Evaluating Performance

Let's look at the number of rules and number of items per rule.

```r
summary(groceryrules)
```

```
## set of 5668 rules
##
## rule length distribution (lhs + rhs):sizes
##    2    3    4    5    6
##   11 1461 3211  939   46
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.00    4.00    3.92    4.00    6.00
##
## summary of quality measures:
##     support           confidence          lift            count
##  Min.   :0.001017   Min.   :0.5000   Min.   : 1.957   Min.   : 10.0
##  1st Qu.:0.001118   1st Qu.:0.5455   1st Qu.: 2.464   1st Qu.: 11.0
##  Median :0.001322   Median :0.6000   Median : 2.899   Median : 13.0
##  Mean   :0.001668   Mean   :0.6250   Mean   : 3.262   Mean   : 16.4
##  3rd Qu.:0.001729   3rd Qu.:0.6842   3rd Qu.: 3.691   3rd Qu.: 17.0
##  Max.   :0.022267   Max.   :1.0000   Max.   :18.996   Max.   :219.0
##
## mining info:
##       data ntransactions support confidence
##  groceries          9835   0.001        0.5
```

Let's see what we found:

2-itemset: 2 rules

3-itemset: 1,461 rules

4-itemset: 3,211 rules

5-itemset: 939 rules

6-itemset: 46 rules

Total = 5,668 rules. Whew!

Let's look at the first 10 rules. Please note the rules are not listed in the order of importance.

```r
inspect(groceryrules[1:10])
```

```
##      lhs                    rhs                support     confidence
## [1]  {honey}             => {whole milk}       0.001118454 0.7333333
## [2]  {tidbits}           => {rolls/buns}       0.001220132 0.5217391
## [3]  {cocoa drinks}      => {whole milk}       0.001321810 0.5909091
## [4]  {pudding powder}    => {whole milk}       0.001321810 0.5652174
## [5]  {cooking chocolate} => {whole milk}       0.001321810 0.5200000
## [6]  {cereals}           => {whole milk}       0.003660397 0.6428571
## [7]  {jam}               => {whole milk}       0.002948653 0.5471698
## [8]  {specialty cheese}  => {other vegetables} 0.004270463 0.5000000
## [9]  {rice}              => {other vegetables} 0.003965430 0.5200000
## [10] {rice}              => {whole milk}       0.004677173 0.6133333
##      lift     count
## [1]  2.870009 11
## [2]  2.836542 12
## [3]  2.312611 13
## [4]  2.212062 13
## [5]  2.035097 13
## [6]  2.515917 36
## [7]  2.141431 29
## [8]  2.584078 42
## [9]  2.687441 39
## [10] 2.400371 46
```

Let's Talk About "Lift"!

In the context of our current analysis, lift measures "how much more likely an item is to be purchased relative to its typical purchase rate, given that you know another item has been purchased" (Lantz 2013, p. 261).

For example:

```
Lift (honey --> whole milk) = Confidence (honey --> whole milk)/Support (whole milk)

Confidence (honey --> whole milk) = 0.7333

Support (whole milk) = 0.2556.

Lift = 0.4108/0.2556 = 2.87
```

Improving Performance

Let's sort the rules by lift.

```r
groceryrules_sorted <- sort(groceryrules, by = "lift")
#inspect(groceryrules_sorted)
```

And now by lift and confidence.

```r
groceryrules_sorted <-sort(groceryrules, by = c("lift", "confidence"))
#inspect(groceryrules_sorted)
```

Strong Rules. Actionable Rules.

A **strong** rule has both high support and confidence.

An **actionable** rule is one you can act on. Remember that there are always more trivial rules than non-trivial, actionable rules.

An Example: It is Soup Season!

Here we are looking at the subsets of rules containing "soups" items. Winter is approaching, and we know people buy soup during colder months. What else are they buying with soup?

```
soups_rules <- subset(groceryrules_sorted, items %in% "soups")
#inspect(soups_rules)
```

Let's write the rules out to a CSV file.

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

Looking at the rules in a data frame.

```
groceryrules_df <- as(groceryrules_sorted, "data.frame")
#View(groceryrules_df)
```

Using arulesViz Package to Visualize the "Mined" Rules

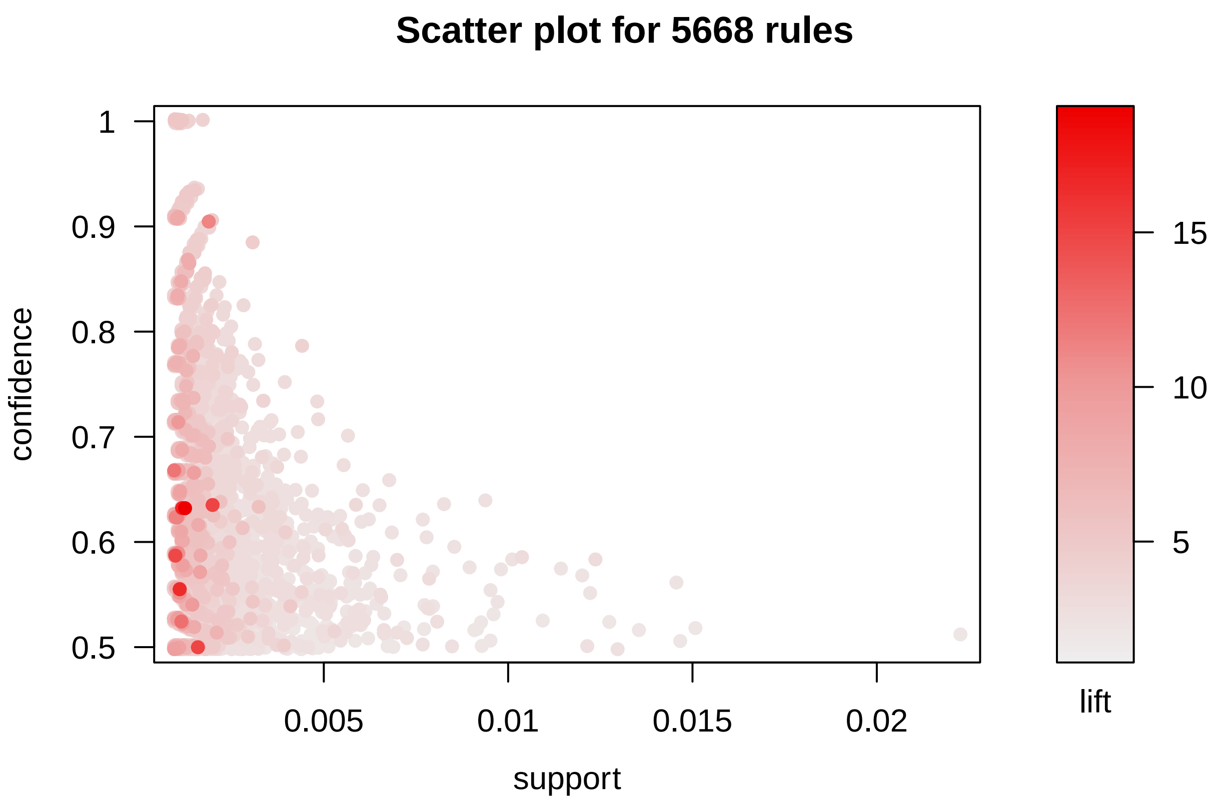
Scatterplot

```
library(arulesViz)
```

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

```
plot(groceryrules)
```

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

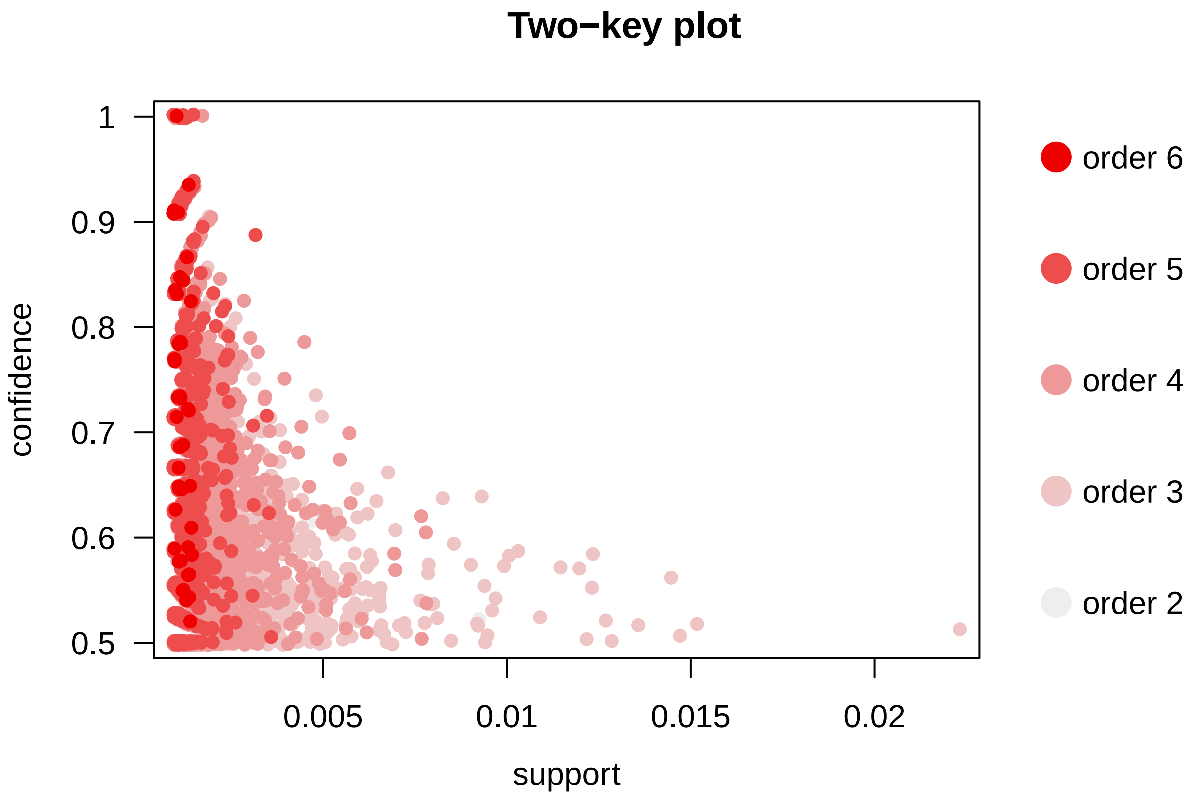


##A two-key plot

Looking at the k-itemset rules by different coding colors.

```
plot(groceryrules, shading="order", control=list(main="Two-key plot"))
```

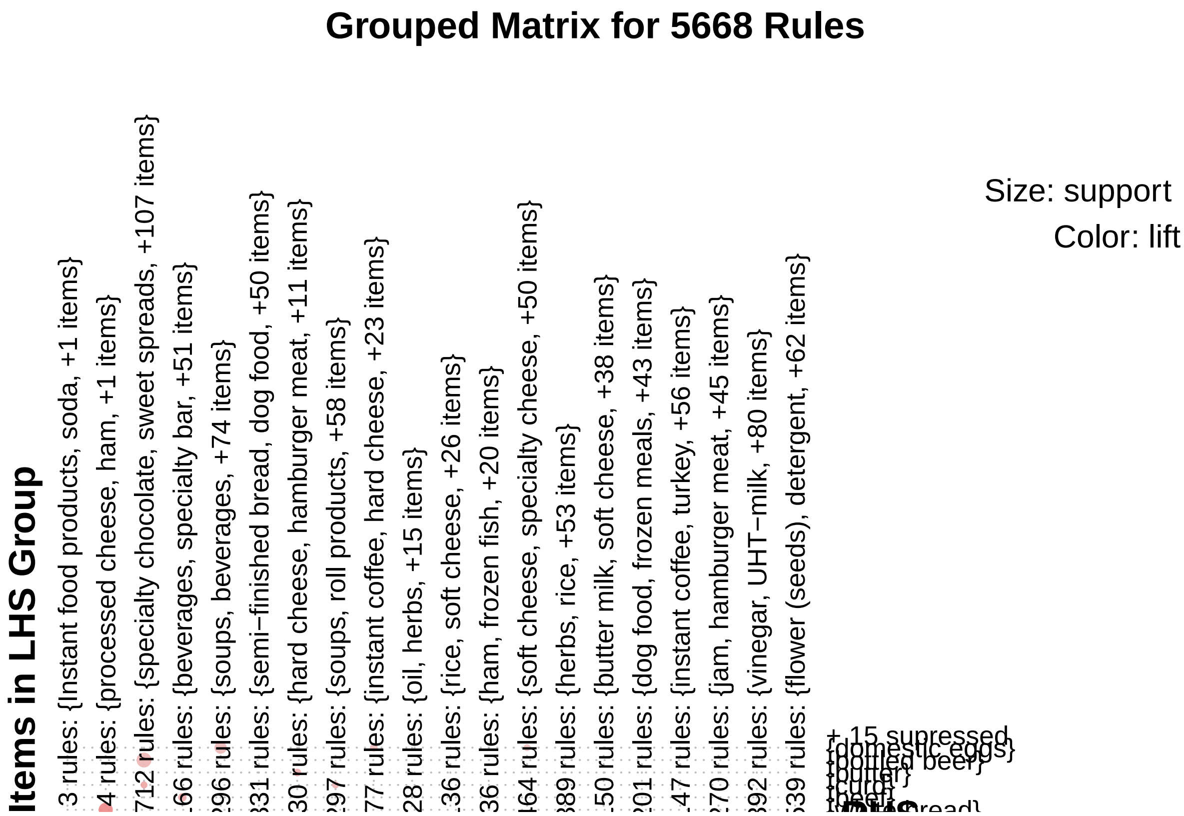
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Grouped Matrix Plot

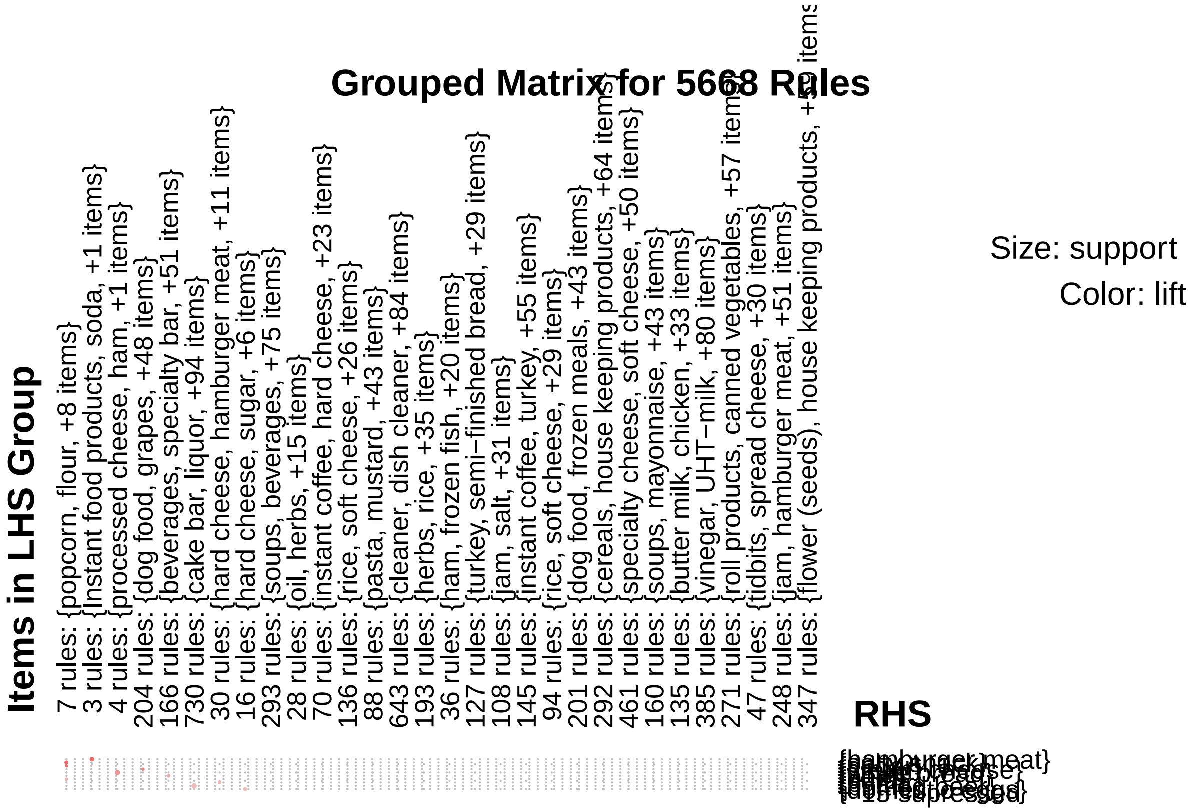
The rules are grouped using k-means clustering. Default quality measure is lift. Default plot shows 20 rules for the antecedents (LHS or left hand side).

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



Let's try 30 rules in LHS

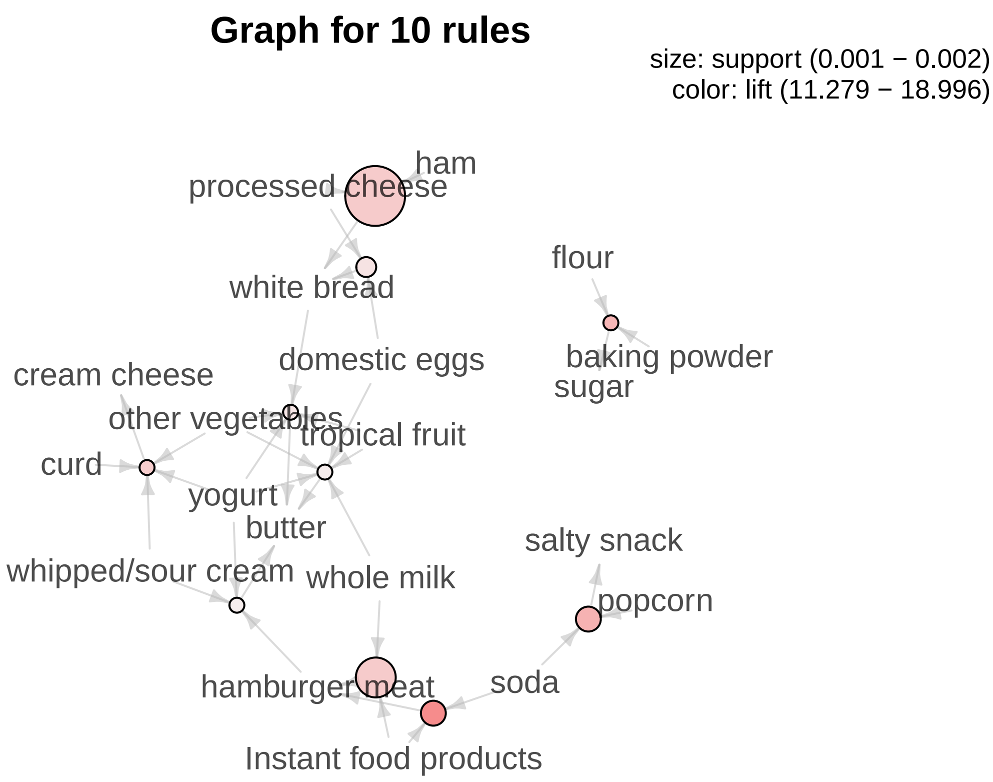
```r
plot(groceryrules, method="grouped", control=list(k=30))
```



Graph Based Visualizations

This technique only works well for a small number of rules. We will create a graph for the first ten rules. Please note that we are using the sorted grocery rules vector. The default setting will give items and their relationships to each other.

```
plot(groceryrules_sorted[1:10], method="graph")
```

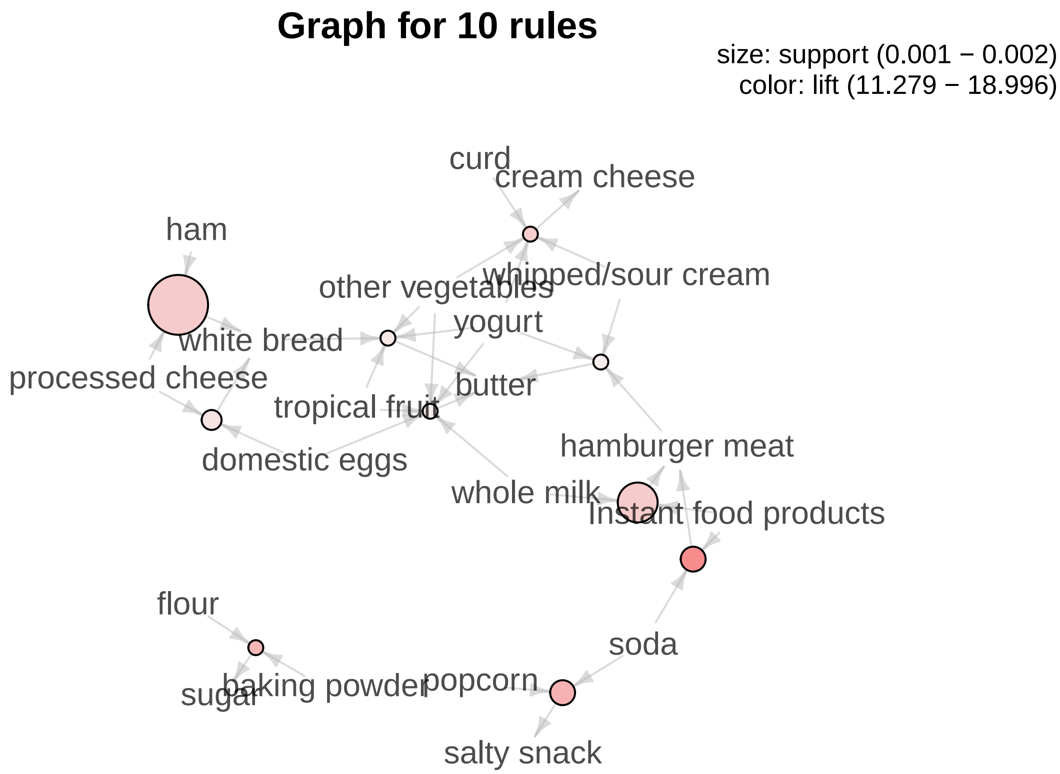


This is another graph type using the itemsets instead.

```
plot(groceryrules_sorted[1:10], method="graph", control=list(type="itemsets"))
```

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

```
## Available control parameters (with default values):
## main  =  Graph for 10 rules
## nodeColors    =  c("#66CC6680", "#9999CC80")
## nodeCol   =  c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF", "#EE0F0FFF", "#EE1212
## edgeCol   =  c("#474747FF", "#494949FF", "#4B4B4BFF", "#4D4D4DFF", "#4F4F4FFF", "#515151FF", "#535353
## 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
```



Mining Rules Interactively

The features are clunky but still useable. Click "end" to leave interactive mode.

```
#plot(groceryrules, interactive=TRUE)
#plot(groceryrules, method="grouped", interactive=TRUE)
```

ECLAT Algorithm

Pro: Not as computationally intensive as Apriori

Con: Works best on smaller datasets

Required data format: Vertical

Support of a 1-itemset is the size of its tidset. Support of k-itemset is the intersection of the tidsets of the corresponding itemsets. For example, the support for {beer, diapers} is counted by matching up the tidsets of beer and diapers.

We can see that as the size of the transaction database increases, it is more advantageous to count the tidsets than to pass through the databases multiple times like in Apriori.

Method

1. Generate 1-itemset candidate and count support at the same time
2. Prune candidates based on minsup threshold
3. Repeat Steps 1 & 2 until no more candidates can be generated or no frequent itemset is found

Grocery Shopping in Belgium

We will use a dataset containing 88,162 grocery receipts from an anonymous Belgian supermarket. The receipts contained 16,470 SKUs. The data was collected between 1999 and 2000. More information about this dataset is available from here.

Source of dataset:

Brijs T., Swinnen G., Vanhoof K., and Wets G. (1999), The use of association rules for product assortment decisions: a case study, in: Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining, San Diego (USA), August 15-18, pp. 254-260. ISBN: 1-58113-143-7.

We will download the dataset into R using a hyperlink.

```
retail <- read.transactions(file="http://fimi.ua.ac.be/data/retail.dat", sep = " ")

summary(retail)
```

```
## transactions as itemMatrix in sparse format with
##  88162 rows (elements/itemsets/transactions) and
##  16470 columns (items) and a density of 0.0006257289
##
## most frequent items:
##      39      48      38      32      41 (Other)
##   50675   42135   15596   15167   14945  770058
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3016 5516 6919 7210 6814 6163 5746 5143 4660 4086 3751 3285 2866 2620 2310
##   16   17   18   19   20   21   22   23   24   25   26   27   28   29   30
## 2115 1874 1645 1469 1290 1205  981  887  819  684  586  582  472  480  355
##   31   32   33   34   35   36   37   38   39   40   41   42   43   44   45
##  310  303  272  234  194  136  153  123  115  112   76   66   71   60   50
##   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60
##   44   37   37   33   22   24   21   21   10   11   10    9   11    4    9
##   61   62   63   64   65   66   67   68   71   73   74   76
##    7    4    5    2    2    5    3    3    1    1    1    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    4.00    8.00   10.31   14.00   76.00
##
## includes extended item information - examples:
##   labels
## 1      0
## 2      1
## 3     10
```

Some questions for us to answer:

1. How many items does a typical receipt contain?
2. Which SKUs appear most frequently?
3. What's the density of the sparse matrix?

```r
freq_retail <- as.data.frame(itemFrequency(retail))
#View(freq_groceries_data_frame)
```

Sort the data frame above and find the support for the most popular SKUs.

Default parameter settings for ECLAT: supp = 0.1 and maxlen = 5

What support and maxlen should we use? Many SKUs have support around 0.01. The mean number of items in receipt is 10.

```r
retail.rules<-eclat(retail, parameter=list(supp=0.01, maxlen=10))
```

```
## Eclat
##
## parameter specification:
##  tidLists support minlen maxlen            target   ext
##     FALSE    0.01      1     10 frequent itemsets FALSE
##
## algorithmic control:
##  sparse sort verbose
##       7   -2    TRUE
##
## Absolute minimum support count: 881
##
## create itemset ...
## set transactions ...[16470 item(s), 88162 transaction(s)] done [0.13s].
## sorting and recoding items ... [70 item(s)] done [0.01s].
## creating sparse bit matrix ... [70 row(s), 88162 column(s)] done [0.01s].
## writing  ... [159 set(s)] done [0.04s].
## Creating S4 object  ... done [0.00s].
```

```r
print(retail.rules)
```

```
## set of 159 itemsets
```

159 rules found! Let's dig in deeper. We can use inspect() to view all the rules.

```r
inspect(retail.rules)
```

```
##       items          support    count
## [1]   {37,38}        0.01186452  1046
## [2]   {286,38}       0.01265852  1116
## [3]   {12925,39}     0.01063950   938
## [4]   {1146,39}      0.01114993   983
## [5]   {39,79}        0.01260180  1111
## [6]   {48,79}        0.01012908   893
## [7]   {1327,39}      0.01311223  1156
## [8]   {1327,48}      0.01097979   968
## [9]   {39,438}       0.01429187  1260
## [10]  {438,48}       0.01162632  1025
## [11]  {39,60}        0.01114993   983
## [12]  {255,39}       0.01198929  1057
```

```
## [13]  {255,48}        0.01198929  1057
## [14]  {39,533}        0.01045802   922
## [15]  {270,39}        0.01354325  1194
## [16]  {270,48}        0.01085502   957
## [17]  {2238,39}       0.01459813  1287
## [18]  {2238,48}       0.01083233   955
## [19]  {110,38,39,48}  0.01169438  1031
## [20]  {110,38,39}     0.01973639  1740
## [21]  {110,38,48}     0.01543749  1361
## [22]  {110,39,48}     0.01176244  1037
## [23]  {110,39}        0.01995191  1759
## [24]  {110,48}        0.01565300  1380
## [25]  {110,38}        0.03090901  2725
## [26]  {147,39}        0.01289671  1137
## [27]  {147,48}        0.01175109  1036
## [28]  {271,39}        0.01626551  1434
## [29]  {271,48}        0.01236360  1090
## [30]  {39,413}        0.01281731  1130
## [31]  {413,48}        0.01287403  1135
## [32]  {36,38,39,48}   0.01225018  1080
## [33]  {36,38,39}      0.02206166  1945
## [34]  {36,38,48}      0.01542615  1360
## [35]  {36,39,48}      0.01265852  1116
## [36]  {36,39}         0.02310519  2037
## [37]  {36,48}         0.01606134  1416
## [38]  {36,38}         0.03164629  2790
## [39]  {39,475,48}     0.01238629  1092
## [40]  {39,475}        0.01701413  1500
## [41]  {475,48}        0.01619745  1428
## [42]  {170,38,39,48}  0.01353191  1193
## [43]  {170,38,39}     0.02290102  2019
## [44]  {170,38,48}     0.01744516  1538
## [45]  {170,39,48}     0.01367936  1206
## [46]  {170,39}        0.02335473  2059
## [47]  {170,48}        0.01766067  1557
## [48]  {170,38}        0.03437989  3031
## [49]  {101,39,48}     0.01073025   946
## [50]  {101,39}        0.01587986  1400
## [51]  {101,48}        0.01487035  1311
## [52]  {310,39,48}     0.01527869  1347
## [53]  {310,39}        0.02100678  1852
## [54]  {310,48}        0.01919194  1692
## [55]  {237,39,48}     0.01411039  1244
## [56]  {237,39}        0.02188018  1929
## [57]  {237,48}        0.01907851  1682
## [58]  {225,39,48}     0.01587986  1400
## [59]  {225,39}        0.02666682  2351
## [60]  {225,48}        0.01969102  1736
## [61]  {39,48,89}      0.02410336  2125
## [62]  {39,89}         0.03118123  2749
## [63]  {48,89}         0.03173703  2798
## [64]  {39,48,65}      0.02038293  1797
## [65]  {39,65}         0.03161226  2787
## [66]  {48,65}         0.02868583  2529
```

```
## [67]  {41,65}         0.01128604   995
## [68]  {32,38,39,48}   0.01401965  1236
## [69]  {32,38,39}      0.02087067  1840
## [70]  {32,38,48}      0.01867018  1646
## [71]  {38,39,41,48}   0.02258343  1991
## [72]  {38,39,41}      0.03460675  3051
## [73]  {38,41,48}      0.02692770  2374
## [74]  {38,39,48}      0.06921349  6102
## [75]  {38,39}         0.11734080 10345
## [76]  {38,48}         0.09010685  7944
## [77]  {38,41}         0.04420272  3897
## [78]  {32,38}         0.03213403  2833
## [79]  {32,39,41,48}   0.01867018  1646
## [80]  {32,39,41}      0.02675756  2359
## [81]  {32,41,48}      0.02340010  2063
## [82]  {32,39,48}      0.06127356  5402
## [83]  {32,39}         0.09590300  8455
## [84]  {32,48}         0.09112770  8034
## [85]  {32,41}         0.03625145  3196
## [86]  {39,41,48}      0.08355074  7366
## [87]  {39,41}         0.12946621 11414
## [88]  {41,48}         0.10228897  9018
## [89]  {39,48}         0.33055058 29142
## [90]  {39}            0.57479413 50675
## [91]  {48}            0.47792700 42135
## [92]  {41}            0.16951748 14945
## [93]  {32}            0.17203557 15167
## [94]  {38}            0.17690161 15596
## [95]  {65}            0.05072480  4472
## [96]  {89}            0.04352215  3837
## [97]  {225}           0.03694335  3257
## [98]  {237}           0.03439123  3032
## [99]  {310}           0.02942311  2594
## [100] {101}           0.02537374  2237
## [101] {170}           0.03515120  3099
## [102] {475}           0.02457975  2167
## [103] {36}            0.03330233  2936
## [104] {413}           0.02132438  1880
## [105] {271}           0.02375173  2094
## [106] {147}           0.02017876  1779
## [107] {110}           0.03169166  2794
## [108] {2238}          0.01945283  1715
## [109] {9}             0.01556226  1372
## [110] {270}           0.01966834  1734
## [111] {185}           0.01560763  1376
## [112] {533}           0.01686668  1487
## [113] {255}           0.01671922  1474
## [114] {60}            0.01688936  1489
## [115] {438}           0.02113155  1863
## [116] {1327}          0.02025816  1786
## [117] {201}           0.01285134  1133
## [118] {79}            0.01814841  1600
## [119] {14098}         0.01464350  1291
## [120] {301}           0.01365668  1204
```

```
## [121] {604}          0.01371339  1209
## [122] {123}          0.01476827  1302
## [123] {338}          0.01445067  1274
## [124] {249}          0.01315760  1160
## [125] {1393}         0.01316894  1161
## [126] {592}          0.01391756  1227
## [127] {117}          0.01163767  1026
## [128] {1146}         0.01617477  1426
## [129] {548}          0.01289671  1137
## [130] {824}          0.01372473  1210
## [131] {12925}        0.01663982  1467
## [132] {783}          0.01094576   965
## [133] {1004}         0.01249972  1102
## [134] {16010}        0.01492707  1316
## [135] {677}          0.01259046  1110
## [136] {589}          0.01269254  1119
## [137] {49}           0.01270389  1120
## [138] {3270}         0.01077562   950
## [139] {10515}        0.01000431   882
## [140] {179}          0.01132007   998
## [141] {19}           0.01139947  1005
## [142] {258}          0.01119530   987
## [143] {522}          0.01104784   974
## [144] {78}           0.01202332  1060
## [145] {479}          0.01050339   926
## [146] {15832}        0.01296477  1143
## [147] {16217}        0.01322565  1166
## [148] {956}          0.01033325   911
## [149] {740}          0.01339579  1181
## [150] {31}           0.01043533   920
## [151] {13041}        0.01192124  1051
## [152] {2958}         0.01025385   904
## [153] {264}          0.01015177   895
## [154] {175}          0.01100247   970
## [155] {286}          0.01341848  1183
## [156] {161}          0.01145618  1010
## [157] {37}           0.01218212  1074
## [158] {45}           0.01033325   911
## [159] {242}          0.01033325   911
```

Alternatively, we can use ruleInduction() and set the confidence level to filter the rules. The default confidence=0.8

```r
retail.rules.review<-ruleInduction(retail.rules,retail)
retail.rules.sorted<-sort(retail.rules.review, by = c("lift", "confidence"))
inspect(retail.rules.sorted)
```

```
##      lhs             rhs  support    confidence lift     itemset
## [1]  {110,39,48} => {38} 0.01169438 0.9942141  5.620153 19
## [2]  {170,39,48} => {38} 0.01353191 0.9892206  5.591925 42
## [3]  {110,39}    => {38} 0.01973639 0.9891984  5.591800 20
## [4]  {170,48}    => {38} 0.01744516 0.9877970  5.583878 44
## [5]  {110,48}    => {38} 0.01543749 0.9862319  5.575030 21
```

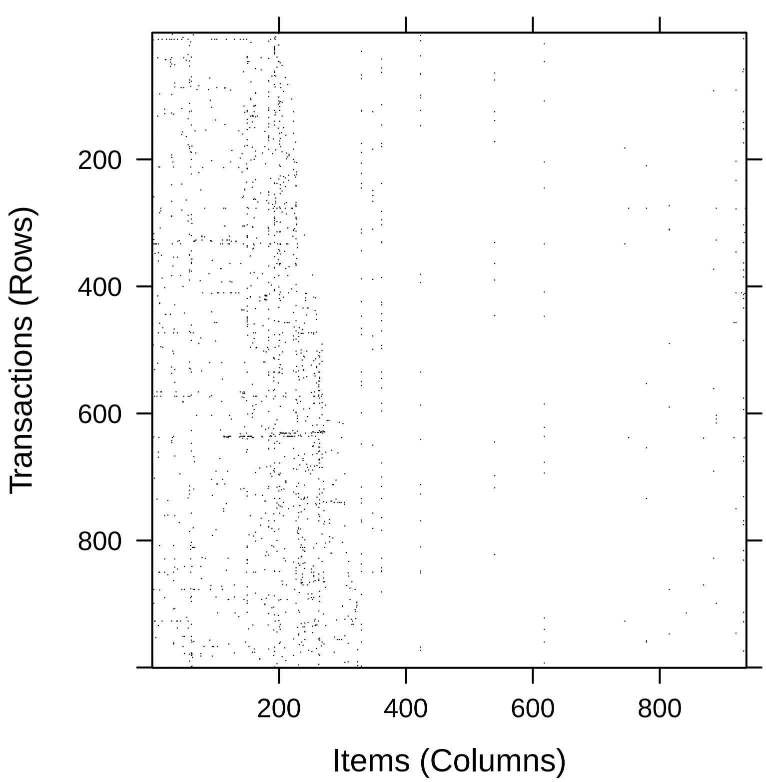
```
## [6]  {170,39}    => {38} 0.02290102 0.9805731  5.543042 43
## [7]  {170}       => {38} 0.03437989 0.9780574  5.528821 48
## [8]  {110}       => {38} 0.03090901 0.9753042  5.513258 25
## [9]  {37}        => {38} 0.01186452 0.9739292  5.505485  1
## [10] {36,39,48}  => {38} 0.01225018 0.9677419  5.470509 32
## [11] {36,48}     => {38} 0.01542615 0.9604520  5.429300 34
## [12] {36,39}     => {38} 0.02206166 0.9548355  5.397551 33
## [13] {36}        => {38} 0.03164629 0.9502725  5.371757 38
## [14] {286}       => {38} 0.01265852 0.9433643  5.332706  2
## [15] {38,41,48}  => {39} 0.02258343 0.8386689  1.459077 71
## [16] {41,48}     => {39} 0.08355074 0.8168108  1.421049 86
## [17] {225,48}    => {39} 0.01587986 0.8064516  1.403027 58
```

Classifying Documents

The Epub dataset is available in the arules package. According to the arules manual documentation, it states, "The Epub dataset contains the download history of documents from the electronic platform of the Vienna University of Economics and Business Administration. The data was recorded between Jan 2003 and Dec 2008. . . .There are 15,729 transactions and 936 items. The item labels are document IDs" (Hahsler 2016, p. 27). The dataset was donated by Michael Hahsler from ePub-WU. Link to the arules documentation

```
data("Epub")

image(Epub[1:1000])
```



```r
freq_Epub <- as.data.frame(itemFrequency(Epub))

Epub.rules<-eclat(Epub,parameter=list(supp=0.001))
```

```
## Eclat
##
## parameter specification:
##  tidLists support minlen maxlen            target   ext
##     FALSE   0.001      1     10 frequent itemsets FALSE
##
## algorithmic control:
##  sparse sort verbose
##       7   -2    TRUE
##
## Absolute minimum support count: 15
##
## create itemset ...
## set transactions ...[936 item(s), 15729 transaction(s)] done [0.00s].
## sorting and recoding items ... [481 item(s)] done [0.00s].
## creating sparse bit matrix ... [481 row(s), 15729 column(s)] done [0.00s].
## writing  ... [561 set(s)] done [0.03s].
## Creating S4 object  ... done [0.00s].
```

```r
summary(Epub.rules)
```

```
## set of 561 itemsets
##
## most frequent items:
## doc_4c7 doc_6bf  doc_71 doc_364 doc_3ec (Other)
##      10      10       8       7       7     600
##
## element (itemset/transaction) length distribution:sizes
##   1   2   3
## 481  79   1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.144   1.000   3.000
##
## summary of quality measures:
##     support             count
##  Min.   :0.001017   Min.   : 16.00
##  1st Qu.:0.001272   1st Qu.: 20.00
##  Median :0.001780   Median : 28.00
##  Mean   :0.002758   Mean   : 43.38
##  3rd Qu.:0.002861   3rd Qu.: 45.00
##  Max.   :0.022633   Max.   :356.00
##
## includes transaction ID lists: FALSE
##
## mining info:
##  data ntransactions support
##  Epub         15729   0.001
```

```r
print(Epub.rules)
```

```
## set of 561 itemsets
```

```r
#inspect(Epub.rules[1:100])
```