R Notebook: Market Basket Analysis

Contents

The following packages are needed.

- arules
- arulesViz
- grid
- car
- plotly

Marketers often want to extract insight from transactions. Association rule analysis attempts to find sets of informative patterns from large, sparse data sets. We demonstrate association rules using a real data set of more than 80,000 market basket transactions with 16,000 unique items.

We then examine how rule mining is potentially useful with transactional data and we use association rules to explore patterns in this transactional data.

1 The Basics of Association Rules

The basic idea of association rule mining is that when events occur together more often than one would expect from their individual rates of occurrence, such co-occurrence is an interesting pattern.

There are some terms required to understand association rules.

- An *association* is simply the co-occurrence of two or more things. Hot dogs might be positively associated with relish, hot dog buns, soda, potato chips, and ketchup. An association is not necessarily strong. In a store such as Costco that sells everything from hot dogs to TVs, everything sold is associated with everything else, but most of those associations are weak. A *set of items* is a group of one or more items and might be written as {item1, item2, . . .}. For instance, a set might be {relish} or {hot dogs, soda, potato chips}.
- A *transaction* contains a set of items that co-occur in an observation. In marketing, a transaction unit is the market basket, the set of things that are purchased or considered for purchase at one time. Any data points that co-occur are considered to be a transaction, even if using the term "transaction" seems unusual in the context. For example, the set of web pages that a user visits during a session would be a transaction in this sense.
- A *rule* expresses the incidence across transactions of one set of items as a condition of another set of items. The association of relish, conditional on hot dogs, is expressed in the rule {relish} $\Rightarrow$ {hot dogs}. That indicates that if customers buy relish, they are also likely to buy hot dogs.

Rules may express the relationship of multiple items; for instance, {relish, ketchup, mustard, potato chips} $\Rightarrow$ {hot dogs, hamburger patties, hot dog buns, soda, beer}. A condition, in this sense, does not imply a causal relationship, only an association of some strength, whether strong or weak.

Association rules are expressed with three common **metrics** that reflect the rules of conditional probability.

1. *Support* for a set of items is the proportion of all transactions that contain the set. If {hot dogs, soda} appears in 10 out of 200 transactions, then *support({hotdogs,soda})* = 0.05. It does not matter if those 10 transactions contain other items; support is defined separately for every unique set of items.
2. *Confidence* is the support for the co-occurrence of all items in a rule, conditional on the support for the left-hand set alone. Thus, *confidence*$(X \Rightarrow Y)$ = *support*$(X \cap Y)$/*support*(X) (where "$\cap$" means "and"). Consider the rule *{relish}* $\Rightarrow$ *{hot dogs}*. If {relish} occurs in 1% of transactions (in other words, *support({relish})=0.01*) and {relish, hot dogs} appears in 0.5%, then *confidence ({relish}* $\Rightarrow$ *{hotdogs}) = 0.005/0.01 = 0.5*. In other words, hot dogs appear alongside relish 50% of the time that relish appears.

3. *Lift* measures the support of a set conditional on the joint support of each element, or lift(X $\Rightarrow$ Y)=support(X $\cap$ Y)/(support(X)support(Y)). To continue the hot dog example, if support({relish})=0.01, support({hotdogs})= 0.01, and support({relish,hotdogs}) = 0.005, then lift({relish $\Rightarrow$ hotdogs}) = 0.005/(0.01 * 0.01) = 50. In other words, the combination {relish, hot dogs} occurs 50 times more often than we would expect if the two items were independent.

These three measures tell us different things. When we search for rules, we wish to exceed a minimum threshold on each: to find item sets that occur relatively frequently in transactions (*support*) that show strong conditional relationships (*confidence*), and that are more common than chance (*lift*).

As we will see, in practice, an analyst sets the level of required support to a value such as 0.01, 0.10, 0.20, or so forth as is meaningful and useful for the business in consideration of the data characteristics (such as the size of the item set). Similarly, the level of required confidence might be high (such as 0.8) or low (such as 0.2) depending on the data and business. For lift, higher values are generally better and certainly should be above 1.0, although one must be mindful of outliers with huge lifts.

2 Retail Transaction Data: Groceries

The dataset we examine contains supermarket transaction data. We first examine a small data set that is included with the *arules* package. This data set is useful despite its small size because the items are labelled with category names, making them easier to read. Then, we turn to a larger data set from a supermarket chain whose data is disguised but is more typical of large data sets.

2.1 Data: *Groceries*

We illustrate the general concepts of association rules with the *Groceries* data set in the *arules* package. This data set comprises lists of items purchased together (that is, market baskets), where the individual items have been recorded as category labels instead of product names. You should install the *arules* and *arulesViz* packages before proceeding.

We load the package and data, and then check the data as follows:

```
library(arules)
```

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

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

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

```
data("Groceries")
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   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   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  level2           level1
## 1 frankfurter sausage meat and sausage
## 2     sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

```
inspect(head(Groceries,3))
```

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
```

We can see that the item sets are structured with brackets, reflect the new "transactions" data type.

2.2 Finding rules

We now use *apriori(data, parameters = ...)* to find association rules with the *apriori* algorithm. At a conceptual level, the apriori algorithm searches through the item sets that frequently occur in a list of transactions. For each item set, it evaluates the various possible rules that express associations among the items at or above a particular level of support, and then retains the rules that show confidence above some threshold value.

To control the extent that *apriori()* searches, we use the *parameter=list()* control to instruct the algorithm to search rules that have a minimum *support* of 0.01 (1% transactions) and extract the ones that further demonstrate a minimum *confidence* of 0.3. The resulting rules set is assigned to the *groc.rules* object:

```
groc.rules <- apriori(Groceries, parameter = list(supp=0.01, conf=0.3, target="rules"))
```

```
## Apriori
##
```

```
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    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 ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

2.3 Inspecting rules

To interpret the results of *apriori()* above, there are two key things to examine.

- First, check the *number of items* going into the rules, which is shown on the output line *"sorting and recoding items ..."* and, in this case, tells us that the rules found are using 88 of the total number of items. If this number is too small (only a tiny set of your items) or too large (almost all of them), then you might wish to adjust the support and confidence levels.
- Next, check the *number of rules* found, as indicated on the *"writing ..."* line. In this case, the algorithm found 125 rules. If this number is too low, it suggests the need to lower the support or confidence levels; if it is too high (such as many more rules than items), you might increase the support or confidence levels.

Once we have a rule set from *apriori()*, we use *inspect(rules)* to examine the association rules. The complete list of 125 from above is too long to examine here, so we select a subset of them with high lift, *lift* > 3. We find that five of the rules in our set have lift greater than 3.0:

```
inspect(subset(groc.rules, lift > 3))
```

```
##     lhs                                rhs                support
## [1] {beef}                          => {root vegetables}  0.01738688
## [2] {citrus fruit,root vegetables}  => {other vegetables} 0.01037112
## [3] {citrus fruit,other vegetables} => {root vegetables}  0.01037112
## [4] {tropical fruit,root vegetables}  => {other vegetables} 0.01230300
## [5] {tropical fruit,other vegetables} => {root vegetables}  0.01230300
##     confidence coverage   lift     count
## [1] 0.3313953  0.05246568 3.040367 171
## [2] 0.5862069  0.01769192 3.029608 102
## [3] 0.3591549  0.02887646 3.295045 102
## [4] 0.5845411  0.02104728 3.020999 121
## [5] 0.3427762  0.03589222 3.144780 121
```

The first rule tells us that if a transaction contains {beef} then it is also relatively more likely to contain {root vegetables} - a category that we assume includes items such as potatoes and onions. The *support* shows that the combination appears in 1.7% of baskets, and the *lift* shows that the combination is 3× more likely to occur together than one would expect from the individual rates of incidence alone.

A store might form several insights on the basis of such information. For instance, the store might create a display for potatoes and onions near the beef counter to encourage shoppers who are examining beef to purchase those vegetables or consider recipes with them. It might also suggest putting coupons for beef in the root vegetable area or featuring recipe cards somewhere in the store.

2.4 Plot rules

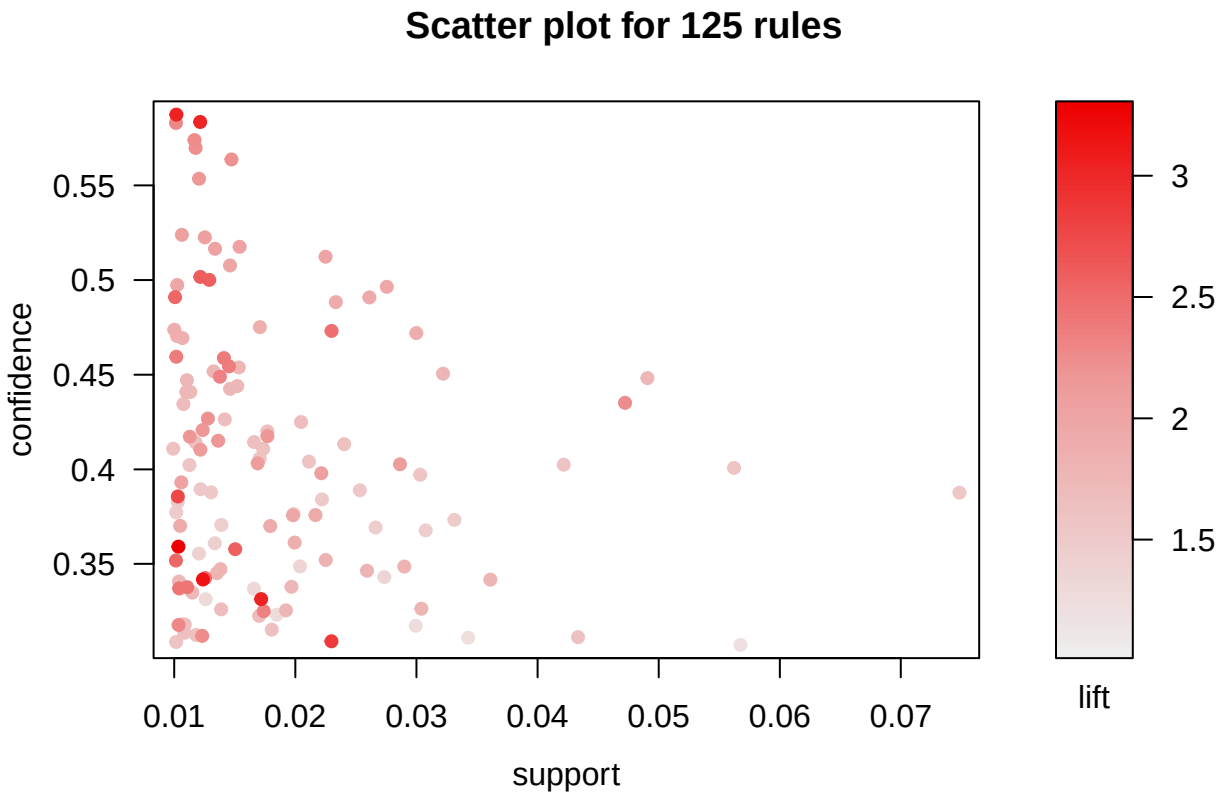
To get a sense of the rule distribution, we load the *arulesViz* package and then *plot()* the rule set, which charts the rules according to confidence (Y axis) by support (X axis) and scales the darkness of points to indicate lift.

```
library(arulesViz)
```

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

```
plot(groc.rules)
```

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



In that chart, we see that most rules involve item combinations that infrequently occur (that is, they have low support) while confidence is relatively smoothly distributed.

Simply showing points is not very useful, and a key feature with *arulesViz* is interactive plotting. In the above figure, there are some rules on the upper left with a high lift. We can use interactive plotting to inspect those rules. To do this, add *interactive=TRUE* to the *plot()* command:

```
library(plotly)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
plot(groc.rules,  engine = "plotly")
```

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

```
## Warning: `arrange_()` is deprecated as of dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
##   PhantomJS   not   found.   You   can   install   it   with   webshot::install_phantomjs().
```

In interactive mode, you can examine regions of rules. To do so, click once in the plot window at one corner of the area of interest and then click again at the opposite corner. You can use *zoom in* to magnify that region or *inspect* to list the rules in the region. When finished, click *end.*

We selected the upper left region and zoomed in on that region. Then, we selected a few rules from the zoomed-in area and clicked inspect to display them in the console. There were seven rules in that subregion. This revealed two high-lift rules.

One rule tells us that the combination {citrus fruit, root vegetables} occurs in about 1.0 % of baskets (support=0.0104), and when it occurs, it highly likely includes {other vegetables} (confidence= 0.586). The combination occurs 3 times more often than we would expect from the individual incidence rates of {citrus fruit, root vegetables} and {other vegetables} considered separately (lift=3.03).

Such information could be used in various ways. If we pair the transactions with customer information, we could use this for targeted mailings or email suggestions. For items often sold together, we could adjust the price and margins together; for instance, to put one item on sale while increasing the price of the other. Or perhaps the cashiers might ask customers, "Would you like some other vegetables with that?"

2.5 Finding and Plotting Subsets of Rules

A common goal in market basket analysis is to find rules with high lift. We can find such rules easily by sorting the larger set of rules by lift. We extract the 15 rules with the highest lift using *sort()* to order the rules by lift and to take 50 from the *head()*:

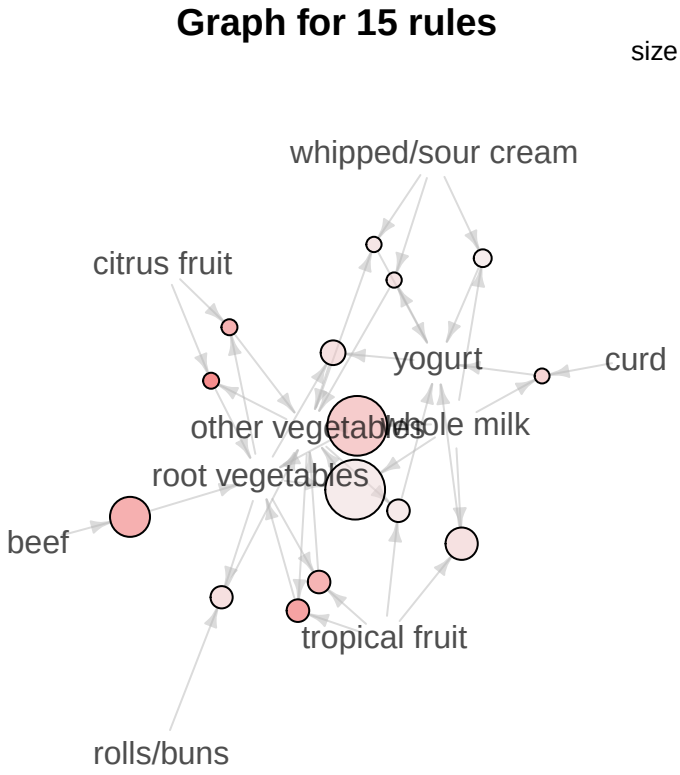
```
groc.hi <- head(sort(groc.rules, by="lift"), 15)
inspect(groc.hi)
```

```
##      lhs                                    rhs                 support
## [1]  {citrus fruit,other vegetables}     => {root vegetables}   0.01037112
## [2]  {tropical fruit,other vegetables}   => {root vegetables}   0.01230300
## [3]  {beef}                              => {root vegetables}   0.01738688
## [4]  {citrus fruit,root vegetables}      => {other vegetables}  0.01037112
## [5]  {tropical fruit,root vegetables}    => {other vegetables}  0.01230300
## [6]  {other vegetables,whole milk}       => {root vegetables}   0.02318251
## [7]  {whole milk,curd}                   => {yogurt}            0.01006609
## [8]  {root vegetables,rolls/buns}        => {other vegetables}  0.01220132
## [9]  {root vegetables,yogurt}            => {other vegetables}  0.01291307
## [10] {tropical fruit,whole milk}         => {yogurt}            0.01514997
## [11] {yogurt,whipped/sour cream}         => {other vegetables}  0.01016777
## [12] {other vegetables,whipped/sour cream} => {yogurt}          0.01016777
## [13] {tropical fruit,other vegetables}   => {yogurt}            0.01230300
## [14] {root vegetables,whole milk}        => {other vegetables}  0.02318251
## [15] {whole milk,whipped/sour cream}     => {yogurt}            0.01087951
##      confidence coverage   lift     count
## [1]  0.3591549  0.02887646 3.295045 102
## [2]  0.3427762  0.03589222 3.144780 121
## [3]  0.3313953  0.05246568 3.040367 171
## [4]  0.5862069  0.01769192 3.029608 102
## [5]  0.5845411  0.02104728 3.020999 121
## [6]  0.3097826  0.07483477 2.842082 228
## [7]  0.3852140  0.02613116 2.761356  99
## [8]  0.5020921  0.02430097 2.594890 120
## [9]  0.5000000  0.02582613 2.584078 127
## [10] 0.3581731  0.04229792 2.567516 149
## [11] 0.4901961  0.02074225 2.533410 100
## [12] 0.3521127  0.02887646 2.524073 100
## [13] 0.3427762  0.03589222 2.457146 121
## [14] 0.4740125  0.04890696 2.449770 228
## [15] 0.3375394  0.03223183 2.419607 107
```

Support and *lift* are identical for an item set regardless of the items' order within a rule (left-hand or right-hand side of the rule). However, *confidence* reflects direction because it computes the occurrence of the right-hand set conditional on the left-hand side set.

A graph display of rules may be useful to seek themes and patterns at a higher level. We chart the top 15 rules by lift with *plot(. . . , method="graph")*:

```
plot(groc.hi, method="graph")
```



The positioning of items on the resulting graph may differ for your system, but the item clusters should be similar. Each circle represents a rule, with inbound arrows coming from items on the left-hand side of the rule and outbound arrows going to the right-hand side. The size (area) of the circle represents the rule's support, and shade represents lift (darker indicates higher lift).

3 Supermarket Data

We now investigate associations in a larger set of retail transaction data from a Belgian supermarket chain. This data set comprises market baskets of items purchased together, where each record includes arbitrarily numbered item numbers without item descriptions (to protect the chain's proprietary data). This data set is made publicly available by Brijs et al.

3.1 Data Preparation

First we use *readLines()* to get the data from where it is hosted:

```
retail.raw <- readLines("retail.dat")
```

We check the *head*, *tail*, and *summary*:

```
head(retail.raw)
```

```
## [1] "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 "
```

```
## [2] "30 31 32 "
## [3] "33 34 35 "
## [4] "36 37 38 39 40 41 42 43 44 45 46 "
## [5] "38 39 47 48 "
## [6] "38 39 48 49 50 51 52 53 54 55 56 57 58 "
```

```
tail(retail.raw)
```

```
## [1] "48 201 255 278 407 479 767 824 986 1395 1598 2022 2283 2375 6725 13334 14006 14099 "
## [2] "39 875 2665 2962 12959 14070 14406 15518 16379 "
## [3] "39 41 101 346 393 413 479 522 586 635 695 799 1466 1786 1994 2449 2830 3035 3591 3722 6217 1149
## [4] "2310 4267 "
## [5] "39 48 2528 "
## [6] "32 39 205 242 1393 "
```

```
summary(retail.raw)
```

```
##    Length     Class      Mode
##     88162 character character
```

Each row in this object represents a single market basket of items purchased together. Within each row, the items have been assigned arbitrary numbers that simply start at 0 and add new item numbers as needed for all later transactions.

The data comprise 88,162 transactions, where the first basket has 30 items (numbered 0–29), the second has 3 items, and so forth. In the tail(), we see that the last market basket had 5 items, most of which—items 32, 39, 205, and 242—have low numbers, reflecting that those particular items first appeared in transactions early in the data set.

In this text format, the data are not ready to use; we must first split each of the transaction text lines into individual items. To do this, we use *strsplit(lines, " ")*. This command splits each line wherever there is a blank space character (" ") and saves the results to a list:

```
retail.list <- strsplit(retail.raw, " ")
```

To label the individual transactions, we assign descriptive names using *names()* and *paste()*:

```
names(retail.list) <- paste("Trans", 1:length(retail.list))
```

We check the data format again, and finally, we remove the *retail.raw* object that is no longer needed:

```
str(retail.list)
```

```
## List of 88162
##  $ Trans 1    : chr [1:30] "0" "1" "2" "3" ...
##  $ Trans 2    : chr [1:3] "30" "31" "32"
##  $ Trans 3    : chr [1:3] "33" "34" "35"
##  $ Trans 4    : chr [1:11] "36" "37" "38" "39" ...
##  $ Trans 5    : chr [1:4] "38" "39" "47" "48"
##  $ Trans 6    : chr [1:13] "38" "39" "48" "49" ...
##  $ Trans 7    : chr [1:6] "32" "41" "59" "60" ...
##  $ Trans 8    : chr [1:3] "3" "39" "48"
```

```
##  $ Trans 9    : chr [1:6] "63" "64" "65" "66" ...
##  $ Trans 10   : chr [1:2] "32" "69"
##  $ Trans 11   : chr [1:4] "48" "70" "71" "72"
##  $ Trans 12   : chr [1:8] "39" "73" "74" "75" ...
##  $ Trans 13   : chr [1:8] "36" "38" "39" "41" ...
##  $ Trans 14   : chr [1:3] "82" "83" "84"
##  $ Trans 15   : chr [1:5] "41" "85" "86" "87" ...
##  $ Trans 16   : chr [1:15] "39" "48" "89" "90" ...
##  $ Trans 17   : chr [1:5] "36" "38" "39" "48" ...
##  $ Trans 18   : chr [1:9] "39" "41" "102" "103" ...
##  $ Trans 19   : chr [1:5] "38" "39" "41" "109" ...
##  $ Trans 20   : chr [1:9] "39" "111" "112" "113" ...
##  $ Trans 21   : chr [1:15] "119" "120" "121" "122" ...
##  $ Trans 22   : chr [1:4] "48" "134" "135" "136"
##  $ Trans 23   : chr [1:15] "39" "48" "137" "138" ...
##  $ Trans 24   : chr [1:4] "39" "150" "151" "152"
##  $ Trans 25   : chr [1:6] "38" "39" "56" "153" ...
##  $ Trans 26   : chr [1:6] "48" "156" "157" "158" ...
##  $ Trans 27   : chr [1:3] "39" "41" "48"
##  $ Trans 28   : chr [1:7] "161" "162" "163" "164" ...
##  $ Trans 29   : chr [1:9] "38" "39" "48" "168" ...
##  $ Trans 30   : chr [1:9] "32" "39" "41" "48" ...
##  $ Trans 31   : chr [1:10] "32" "38" "39" "47" ...
##  $ Trans 32   : chr [1:4] "39" "184" "185" "186"
##  $ Trans 33   : chr [1:7] "36" "38" "41" "48" ...
##  $ Trans 34   : chr [1:15] "39" "48" "186" "189" ...
##  $ Trans 35   : chr [1:10] "39" "201" "202" "203" ...
##  $ Trans 36   : chr [1:9] "39" "65" "193" "210" ...
##  $ Trans 37   : chr [1:10] "179" "216" "217" "218" ...
##  $ Trans 38   : chr [1:3] "225" "226" "227"
##  $ Trans 39   : chr [1:7] "39" "41" "48" "228" ...
##  $ Trans 40   : chr [1:14] "36" "38" "39" "232" ...
##  $ Trans 41   : chr [1:4] "39" "243" "244" "245"
##  $ Trans 42   : chr [1:8] "39" "41" "48" "246" ...
##  $ Trans 43   : chr [1:6] "39" "48" "65" "251" ...
##  $ Trans 44   : chr [1:3] "48" "230" "254"
##  $ Trans 45   : chr [1:12] "39" "48" "66" "78" ...
##  $ Trans 46   : chr [1:3] "39" "48" "262"
##  $ Trans 47   : chr [1:9] "36" "38" "39" "225" ...
##  $ Trans 48   : chr [1:6] "39" "242" "268" "269" ...
##  $ Trans 49   : chr [1:8] "39" "48" "79" "146" ...
##  $ Trans 50   : chr "274"
##  $ Trans 51   : chr [1:13] "32" "38" "39" "48" ...
##  $ Trans 52   : chr [1:3] "39" "48" "68"
##  $ Trans 53   : chr [1:10] "38" "39" "48" "95" ...
##  $ Trans 54   : chr [1:16] "39" "41" "48" "212" ...
##  $ Trans 55   : chr [1:3] "300" "301" "302"
##  $ Trans 56   : chr [1:23] "36" "38" "39" "105" ...
##  $ Trans 57   : chr [1:7] "10" "322" "323" "324" ...
##  $ Trans 58   : chr [1:5] "39" "48" "152" "161" ...
##  $ Trans 59   : chr [1:3] "39" "329" "330"
##  $ Trans 60   : chr [1:10] "48" "331" "332" "333" ...
##  $ Trans 61   : chr [1:14] "18" "37" "38" "41" ...
##  $ Trans 62   : chr [1:7] "32" "39" "41" "48" ...
```

```
##  $ Trans 63   : chr [1:15] "48" "351" "352" "353" ...
##  $ Trans 64   : chr [1:2] "365" "366"
##  $ Trans 65   : chr [1:14] "38" "39" "41" "48" ...
##  $ Trans 66   : chr [1:17] "1" "11" "39" "41" ...
##  $ Trans 67   : chr [1:4] "386" "387" "388" "389"
##  $ Trans 68   : chr [1:3] "38" "41" "390"
##  $ Trans 69   : chr [1:3] "38" "55" "391"
##  $ Trans 70   : chr [1:15] "32" "43" "151" "152" ...
##  $ Trans 71   : chr [1:6] "338" "400" "401" "402" ...
##  $ Trans 72   : chr [1:4] "39" "405" "406" "407"
##  $ Trans 73   : chr [1:22] "48" "89" "101" "179" ...
##  $ Trans 74   : chr [1:8] "39" "45" "48" "248" ...
##  $ Trans 75   : chr [1:7] "141" "344" "427" "428" ...
##  $ Trans 76   : chr [1:4] "39" "432" "433" "434"
##  $ Trans 77   : chr [1:7] "39" "48" "65" "435" ...
##  $ Trans 78   : chr [1:23] "15" "23" "36" "38" ...
##  $ Trans 79   : chr [1:11] "48" "451" "452" "453" ...
##  $ Trans 80   : chr [1:16] "37" "38" "48" "147" ...
##  $ Trans 81   : chr [1:6] "39" "48" "472" "473" ...
##  $ Trans 82   : chr [1:3] "39" "41" "476"
##  $ Trans 83   : chr [1:3] "477" "478" "479"
##  $ Trans 84   : chr [1:9] "39" "161" "480" "481" ...
##  $ Trans 85   : chr [1:7] "32" "39" "41" "48" ...
##  $ Trans 86   : chr [1:6] "38" "39" "41" "105" ...
##  $ Trans 87   : chr [1:2] "60" "381"
##  $ Trans 88   : chr [1:17] "11" "39" "48" "255" ...
##  $ Trans 89   : chr "39"
##  $ Trans 90   : chr [1:3] "41" "110" "501"
##  $ Trans 91   : chr [1:8] "32" "38" "39" "48" ...
##  $ Trans 92   : chr [1:3] "38" "41" "504"
##  $ Trans 93   : chr [1:14] "225" "232" "347" "505" ...
##  $ Trans 94   : chr [1:9] "38" "39" "41" "48" ...
##  $ Trans 95   : chr [1:2] "39" "48"
##  $ Trans 96   : chr [1:4] "38" "39" "281" "517"
##  $ Trans 97   : chr [1:4] "2" "518" "519" "520"
##  $ Trans 98   : chr [1:3] "310" "521" "522"
##  $ Trans 99   : chr [1:3] "41" "523" "524"
##   [list output truncated]
```

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:arules':
##
##     recode
```

```
some(retail.list) #note: random sample; your results may vary
```

```
## $`Trans 10755`
## [1] "4402" "6526"
##
## $`Trans 11351`
##  [1] "38"   "39"   "41"   "48"   "61"   "78"   "92"   "312"  "370"  "533"
## [11] "604"  "664"  "730"  "757"  "793"  "797"  "798"  "809"  "1013" "1154"
## [21] "1196" "1489" "1659" "1834" "2376" "2505" "2792" "2797" "2799" "3053"
## [31] "3106" "3230" "3412" "3565" "3664" "6031" "6585" "6712"
##
## $`Trans 13171`
## [1] "39"   "48"   "789"  "1517" "3246"
##
## $`Trans 14299`
## [1] "260" "647" "769"
##
## $`Trans 21426`
##  [1] "41"   "48"   "197"  "592"  "623"  "809"  "1246" "4321" "5309" "6266"
## [11] "9527"
##
## $`Trans 24292`
##  [1] "39"    "41"    "48"    "270"   "271"   "681"   "1425"  "3601"  "5826"
## [10] "6570"  "7944"  "10812"
##
## $`Trans 31391`
##  [1] "32"    "39"    "41"    "1880"  "2238"  "2420"  "2624"  "2625"  "2696"
## [10] "12032"
##
## $`Trans 36245`
## [1] "36"   "38"   "39"   "41"   "338"  "589"  "812"  "2238"
##
## $`Trans 37046`
##  [1] "52"   "237"  "1052" "1074" "1569" "1570" "1600" "1739" "1814" "3579"
## [11] "4095" "8409" "9541"
##
## $`Trans 70874`
## [1] "38"   "39"   "48"   "185"  "251"  "270"  "390"  "438"  "5152"
```

```
rm(retail.raw)
```

Using *str()*, we confirm that the list has 88,162 entries and that individual entries look appropriate. *some()* samples a few transactions throughout the larger set for additional confirmation.

We then convert it to the *transactions* object, which enhances the ways we can work with the data and speeds up *arules* operations. To convert from a list to transactions, we cast the object using *as(. . . , "transactions")*:

```
retail.trans <- as(retail.list, "transactions") #takes a few seconds
summary(retail.trans)
```

```
## 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   16
## 3016 5516 6919 7210 6814 6163 5746 5143 4660 4086 3751 3285 2866 2620 2310 2115
##   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32
## 1874 1645 1469 1290 1205  981  887  819  684  586  582  472  480  355  310  303
##   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47   48
##  272  234  194  136  153  123  115  112   76   66   71   60   50   44   37   37
##   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63   64
##   33   22   24   21   21   10   11   10    9   11    4    9    7    4    5    2
##   65   66   67   68   71   73   74   76
##    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
##
## includes extended transaction information - examples:
##   transactionID
## 1       Trans 1
## 2       Trans 2
## 3       Trans 3
```

```
rm(retail.list)  #remove retail.list as it is not needed anymore.
```

Looking at the *summary()* of the resulting object, we see that the transaction-by-item matrix is 88,162 rows by 16,470 columns. Of those 1.4 billion intersections, only 0.06% have positive data (*density*) because most items are not purchased in most transactions. Item 39 appears the most frequently and occurs in 50,675 baskets or more than half of all transactions. 3,016 of the transactions contain only a single item ("sizes" = 1) and the median basket size is 8 items.

3.2 Exercise:

Please now use the prepared supermarket data to do the market basket analysis, finding and visualizing the association rules.

4 Key Point

Association rules are a powerful way to explore the relationships in a data set.

- Association rules are commonly used with sparse data sets that have many observations but little information per observation. In marketing, this is typical of market baskets and similar transaction data.

- The *arules* package is the standard R package for association rules. *arules* provides support for handling sparse data and finding rules, and the *arulesViz* package provides visualization methods.
- Core *metrics* for evaluating association rules are *support* (frequency), *confidence* (co-occurrence), and *lift* (co-occurrence above the rate of association by pure chance). There is no absolute value required of them except that lift should be somewhat greater than 1.0. Interpretation depends on experience with similar data and the usefulness of a particular business question.
- A typical workflow for association rules is:
  - Import the raw data and use *as(data, "transactions")* to transform it into a transactions object for better performance.
  - Use *apriori(transactions, support= , confidence= , target="rules")* to find a set of association rules.
  - Plot the resulting rule with *plot(. . . , interactive=TRUE)* and inspect the rules.
  - Look for patterns by selecting subsets of rules, such as those with the highest lift, and use *plot(. . . , method="graph")* for visualization.