Lab 3 - Association Rule Mining of UCI Online Retail Data

Peter Kouvaris, Daniel Freeman, Ireti Fasere, & Tim McWilliams

Business Understanding

Our dataset contains all customer transactions for a UK-based online retailer during the period between January 12, 2010 and September 12, 2011. The company mainly sells unique all-occasion gifts and many of its customers are wholesalers.

A good association rule algorithm will yield itemsets that highlight important patterns in customer purchase behavior. When reviewed by a human with context about the business, these patterns should make logical sense. Association rule mining is a highly subjective process, where human insight is needed to tweak the parameters of the model until outputs are representative of some underlying pattern.

Association rule mining makes sense for transactional data like these because it provides information relevant to making marketing and purchasing decisions. Stakeholders are primarily interested in maximizing profit for their company; this is achieved by either reducing costs or increasing revenues. Our dataset includes large amounts of physical goods purchased at varying volumes over the course of 20 months. Optimizing this problem through association rule mining will give stakeholders insights that may help improve margins.

Data Understanding

Meaning & Variable Type

The header of our raw data file as downloaded from UCI's website[1] is printed in the code blocks below. This is followed by a description of the variables and then data scale is reviewed. The most relevant of these for association rule mining are the unique counts for class variables. Numeric variables will be binned based on percentiles, but range is still relevant and should be noted.

Table 1: Raw Data Header - Cols 1:5

InvoiceNo	StockCode	Description	Quantity	InvoiceDate
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26
536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26

Table 2: Raw Data Header - Cols 6:8

UnitPrice	CustomerID	Country
2.55	17850	United Kingdom
3.39	17850	United Kingdom
2.75	17850	United Kingdom
3.39	17850	United Kingdom
3.39	17850	United Kingdom

[1] UCI Online Retail Data Set: http://archive.ics.uci.edu/ml/datasets/online+retail

The column data types are as follows:

```
 Column Name  Data Type                 Description
-----------------------------------------------------------------------------------
InvoiceNo    | int                     | categorical var dictating transaction instance
StockCode    | string                  | denotes stock sold in line item
Description  | string                  | name for stock, although not 1:1
Quantity     | int                     | quantity of stock purchased
InvoiceDate  | datetime MM/DD/YY H:MM  | ISO format of time item was purchased
UnitPrice    | float                   | price in GBP of 1 quantity
CustomerID   | int                     | categorical var dictating customer
Country      | string                  | country of customer
-----------------------------------------------------------------------------------
```

```python
import pandas as pd
py_df = pd.read_csv('./online_retail.csv')
print("Unique Counts")
for i in py_df.columns:
  count = len(py_df[i].unique())
  print(""" %s - %s """ % (i,str(count)))

print("NaN Counts")
```

```
## Unique Counts
##  InvoiceNo - 25900
##  StockCode - 4070
##  Description - 4224
##  Quantity - 722
##  InvoiceDate - 23260
##  UnitPrice - 1630
##  CustomerID - 4373
##  Country - 38
## NaN Counts
```

Quality, Shape & Scale

The raw data looks to have already been partially pre-processed. The primary source of NAs are in CustomerID, where an identification number is not tied to a transaction. This will be irrelevant to both of our mining reviews and can be left as is.

```python
import pandas as pd
#import numpy as np
import seaborn as sns
df_cpy = pd.read_csv('./online_retail.csv')
sns.distplot((df_cpy['Quantity']))
```

Summary of Dataframe

```r
summary(df)
```

```
##    InvoiceNo        StockCode
##  573585 :  1114   85123A :  2313
##  581219 :   749   22423  :  2203
##  581492 :   731   85099B :  2159
##  580729 :   721   47566  :  1727
```

```
## 558475 :   705   20725  :  1639
## 579777 :   687   84879  :  1502
## (Other):537202   (Other):530366
##                               Description       Quantity
## WHITE HANGING HEART T-LIGHT HOLDER:  2369   Min.   :-80995.00
## REGENCY CAKESTAND 3 TIER          :  2200   1st Qu.:     1.00
## JUMBO BAG RED RETROSPOT           :  2159   Median :     3.00
## PARTY BUNTING                     :  1727   Mean   :     9.55
## LUNCH BAG RED RETROSPOT           :  1638   3rd Qu.:    10.00
## ASSORTED COLOUR BIRD ORNAMENT     :  1501   Max.   : 80995.00
## (Other)                           :530315
##          InvoiceDate       UnitPrice          CustomerID
## 10/31/11 14:41:  1114   Min.   :-11062.06   Min.   :12346
## 12/8/11 9:28  :   749   1st Qu.:     1.25   1st Qu.:13953
## 12/9/11 10:03 :   731   Median :     2.08   Median :15152
## 12/5/11 17:24 :   721   Mean   :     4.61   Mean   :15288
## 6/29/11 15:58 :   705   3rd Qu.:     4.13   3rd Qu.:16791
## 11/30/11 15:13:   687   Max.   : 38970.00   Max.   :18287
## (Other)       :537202                       NA's   :135080
##            Country
## United Kingdom:495478
## Germany       :  9495
## France        :  8557
## EIRE          :  8196
## Spain         :  2533
## Netherlands   :  2371
## (Other)       : 15279
```

Data Processing Methods

Our raw data is most useful for association rule mining when transformed two different ways. The first transformation was created using a *group by* command on InvoiceNo to create transaction baskets. This allowed us to use association algorithms to analyze what pairs of goods are purchased. The second transformation was created using various methods on numeric variables to transform them into categorical so that we explore patterns separate from order content like time, volume and frequency[2].

The header of this data set is shown in the figure below:

Table 3: Processed Data Header - Cols 1:5

StockCode	Description	Country	month_sold	day
85123A	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom	12_	1_
71053	WHITE METAL LANTERN	United Kingdom	12_	1_
84406B	CREAM CUPID HEARTS COAT HANGER	United Kingdom	12_	1_
84029G	KNITTED UNION FLAG HOT WATER BOTTLE	United Kingdom	12_	1_
84029E	RED WOOLLY HOTTIE WHITE HEART.	United Kingdom	12_	1_

Table 4: Processed Data Header - Cols 6:9

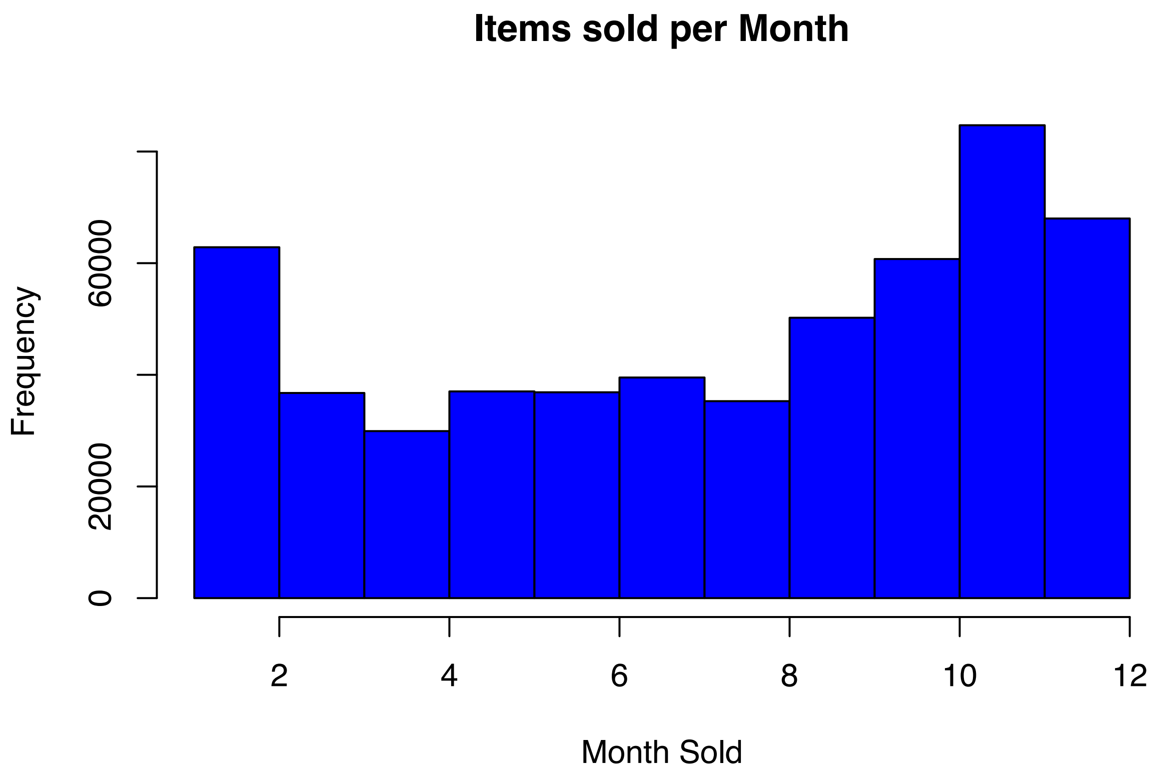
day_of_year	customer_id	quantity_groups	price_groups
335_	17850.0_	Qty_Lvl_3	Price_Lvl_3

[2]https://github.com/htpeter/pdti_DataMining/blob/master/Lab_3/pre-processing.ipynb

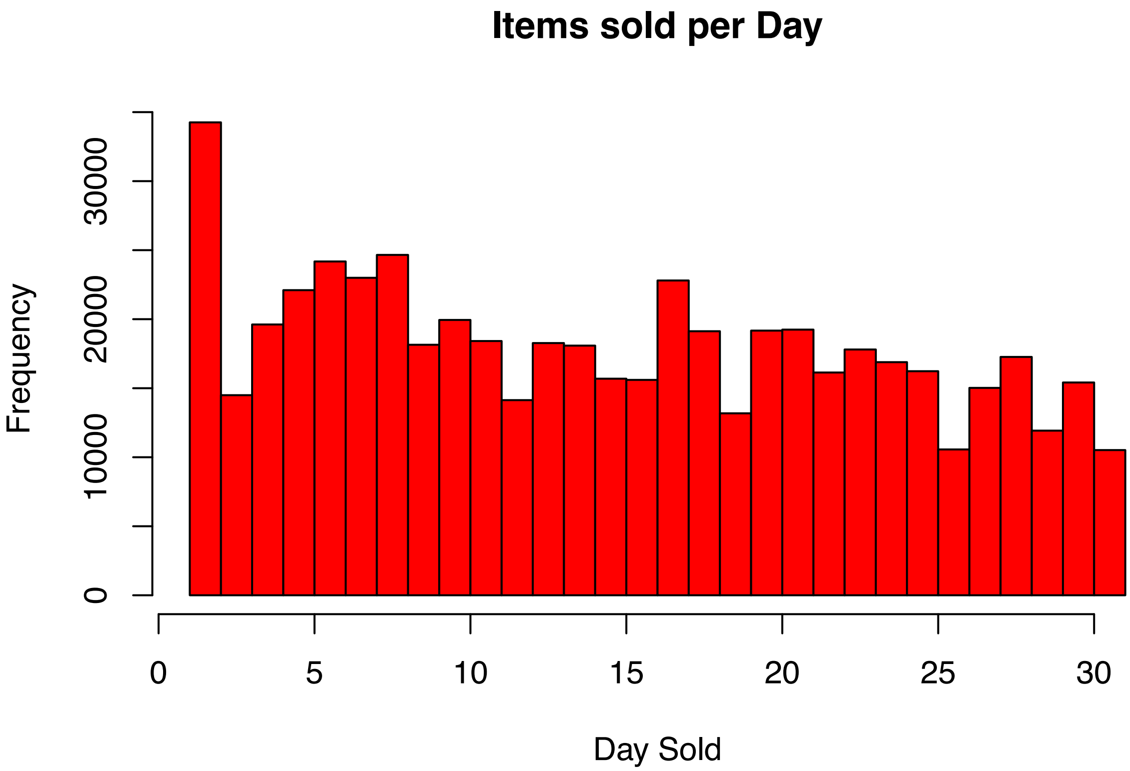
day_of_year	customer_id	quantity_groups	price_groups
335_	17850.0_	Qty_Lvl_3	Price_Lvl_4
335_	17850.0_	Qty_Lvl_4	Price_Lvl_4
335_	17850.0_	Qty_Lvl_3	Price_Lvl_4
335_	17850.0_	Qty_Lvl_3	Price_Lvl_4

Attribute Visualization

When looking at the frequency of items sold per month we can conclude that there are more items sold at the end of the year, specifically October through December. This is because of the holidays. In November people are buying things for Christmas or Hanukah. There is also a high volume of items sold in January. In January, people continue buying items with their money from the holidays.

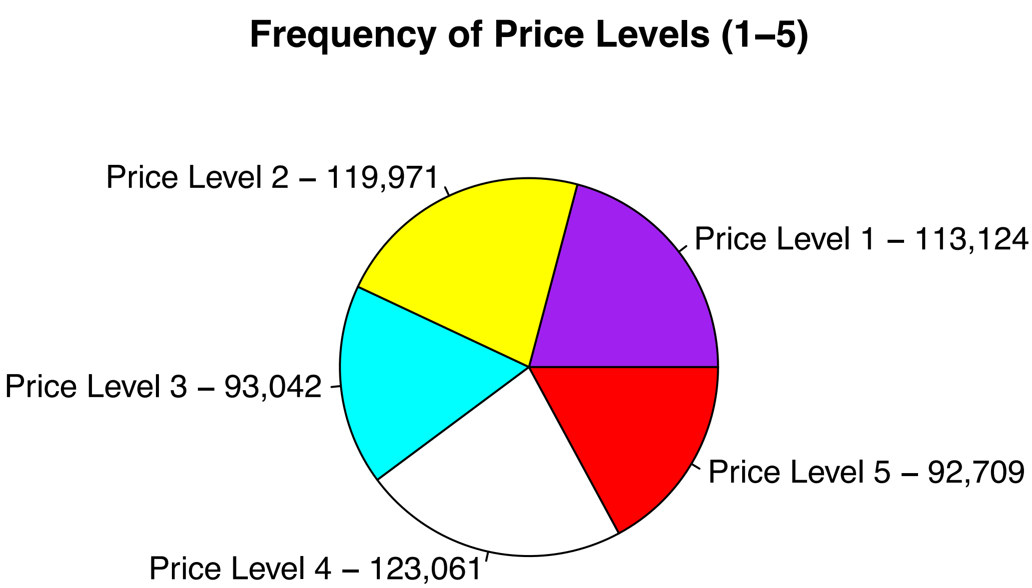


The items sold per day tells a compelling story. More items were sold on the first day of the month and then the number gradually decreases as the month progresses. There were around 34,000 items sold on the first of the month. After the first, the next highest selling day was the seventh at around 25,000 items.



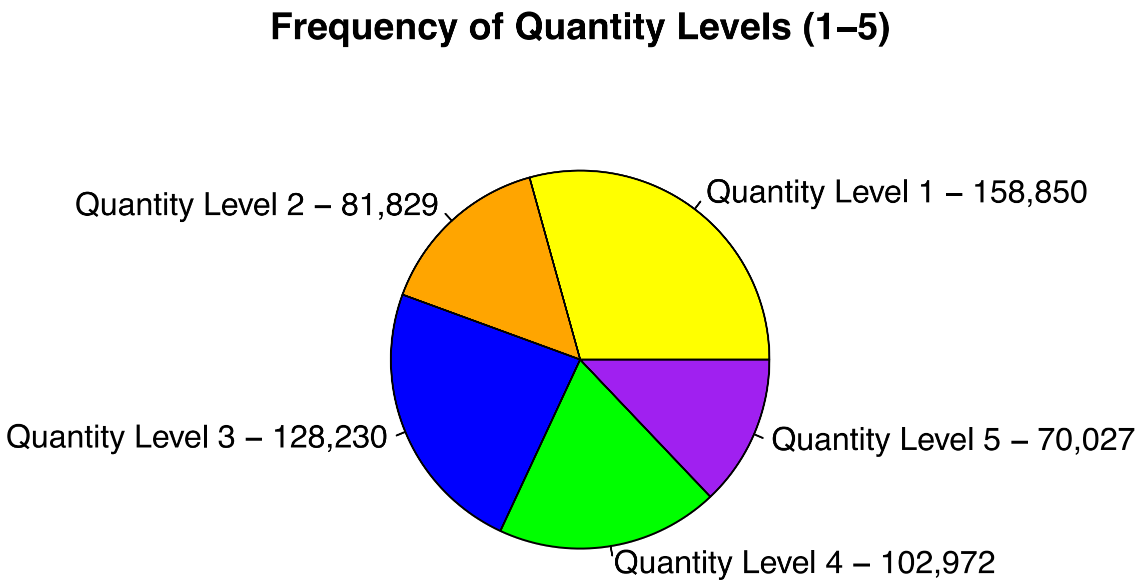
Price Groups

The unit prices were categorized into five different levels. Price level 1 represents the least expensive items, \$0.00 to \$0.86. Price level 2 represents items that are priced between \$0.85 and \$1.65. Price level 3 represents items that are between \$1.65 and \$2.59. Price level 4 represents items that cost between \$2.59 and \$4.95. Price level 5 represents items that are between \$4.95 and \$38,970. Price level 4 had the most items purchased at 123,061 items. Price level 5 had the least items purchased at 92,709 items.



Quantity Groups

```
Quantity Group 1 [0-1]
Quantity Group 2 [2]
Quantity Group 3 [3-6]
Quantity Group 4 [7-13]
Quantity Group 5 [14-80,995]
```



Models

Our models look at two primary measures of interest within our data. The first measure is what types of items are commonly purchased together. The second is how metadata for transactions relate to each other. We find that both these views provide varied business insight. The first highlights what types of goods should be considered related from an inventory and marketing standpoint. The second highlights macro level data that would assist management in navigating yearly trends.

View 1 - Inventory Itemsets Made on Invoice Level

The first analysis will give us information regarding what products are purchsed together. We begin by importing the raw data and creating baskets of transactions. Dplyr is very useful for this.

```
Orig = read.csv('./online_retail.csv',header=TRUE, sep=",",na.strings=c("","NA"))
# Create baskets dataframe
df_baskets <- Orig %>%
group_by(InvoiceNo,InvoiceDate) %>%
summarise(basket = as.vector(list(StockCode)))
#Compute transactions
transactions <- as(df_baskets$basket, "transactions")
```

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

We then review the most common baskets.

```
item_frequencies <- itemFrequency(transactions, type="a")
support <- 0.02
```

```r
freq_items <- sort(item_frequencies, decreasing = F)
freq_items <- freq_items[freq_items>support*length(transactions)]
```

Now that the data base been sorted into baskets where each item purchased at the same time is in the same row, the apriori function can effectively analyze patterns. In our tests we found the below support and confidence parameters to return a narrow set of heavily supported items. We then inspect them to see the general structure of our output.

```r
# run the apriori algorithm
support <- 0.02
itemsets <- apriori(transactions, parameter=list(target= "frequent itemsets",
                                   minlen = 2, support=0.02, conf = 0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##          NA    0.1    1 none FALSE            TRUE       5    0.02      2
##  maxlen            target  ext
##      10 frequent itemsets FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 518
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[4070 item(s), 25943 transaction(s)] done [0.07s].
## sorting and recoding items ... [184 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 done [0.01s].
## writing ... [33 set(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```r
# sort and display frequent itemsets
itemsets <- sort(itemsets, by="support")
inspect(head(itemsets, n=10))
```

```
##      items          support    count
## [1]  {22386,85099B} 0.03207031 832
## [2]  {22697,22699}  0.03022010 784
## [3]  {21931,85099B} 0.02817716 731
## [4]  {22411,85099B} 0.02632695 683
## [5]  {20725,22383}  0.02551748 662
## [6]  {20725,20727}  0.02493929 647
## [7]  {22726,22727}  0.02490074 646
## [8]  {22697,22698}  0.02482365 644
## [9]  {22698,22699}  0.02366727 614
## [10] {20725,22384}  0.02355163 611
```

```r
length(itemsets)
```

```
## [1] 33
```

The above itemsets represent items commonly purchased together. Interestingly, the apriori rule has returned only itemsets with length of 2.

Below we translated these StockCodes to Descriptions.

```
['22386','85099B'] - [JUMBO BAG PINK POLKADOT, JUMBO BAG RED RETROSPOT]
['22697','22699'] - [GREEN REGENCY TEACUP AND SAUCER, ROSES REGENCY TEACUP AND SAUCER ]
['21931','85099B'] - [JUMBO STORAGE BAG SUKI, JUMBO BAG RED RETROSPOT]
['22411','85099B'] - [JUMBO SHOPPER VINTAGE RED PAISLEY, JUMBO BAG RED RETROSPOT]
['20725','22383'] - [LUNCH BAG RED SPOTTY, LUNCH BAG SUKI DESIGN]
['20725','20727'] - [LUNCH BAG RED SPOTTY, LUNCH BAG  BLACK SKULL.]
['22726','22727'] - [ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK BAKELIKE RED ]
['22697','22698'] - [GREEN REGENCY TEACUP AND SAUCER, PINK REGENCY TEACUP AND SAUCER]
['22698','22699'] - [PINK REGENCY TEACUP AND SAUCER, ROSES REGENCY TEACUP AND SAUCER ]
['20725','22384'] - [LUNCH BAG RED RETROSPOT, LUNCH BAG PINK POLKADOT]
```

There is a substantial amount of overlap in purchases where the same items are being purchased in different style variants. For example, a customer bought both a red poka dotted lunch bag and a skull design lunch bag. If we coerce the algorithm to select mininmum length 3 itemsets, will we see this pattern expand?

```r
# run the apriori algorithm
support <- 0.02
itemsets <- apriori(transactions,
          parameter=list(target= "frequent itemsets",
          minlen = 3, support=0.02, conf = 0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##          NA    0.1    1 none FALSE            TRUE       5    0.02      3
##  maxlen            target   ext
##      10 frequent itemsets FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 518
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[4070 item(s), 25943 transaction(s)] done [0.06s].
## sorting and recoding items ... [184 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 done [0.01s].
## writing ... [1 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
# sort and display frequent itemsets
itemsets <- sort(itemsets, by="support")
inspect(head(itemsets, n=10))
```

```
##     items               support    count
## [1] {22697,22698,22699} 0.02116178 549
```

```r
length(itemsets)
```

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

When the itemsets are limited to length of 3, we see only one instance. Not suprisingly, the most supported 2-itemset above is a subset of this 3-itemset.

```
[22697,22698,22699] - GREEN REGENCY TEACUP AND SAUCER,
PINK REGENCY TEACUP AND SAUCER,
ROSES REGENCY TEACUP AND SAUCER
```

```r
# generate some rules from the frequent itemsets
rules <- apriori(transactions, parameter = list(minlen=2,supp=0.02, conf=0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5    0.02      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: 518
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[4070 item(s), 25943 transaction(s)] done [0.07s].
## sorting and recoding items ... [184 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 done [0.01s].
## writing ... [3 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```r
inspect(head(rules, n=10))
```

```
##     lhs              rhs     support    confidence lift     count
## [1] {22698}       => {22697} 0.02482365 0.8029925  19.70864 644
## [2] {22698,22699} => {22697} 0.02116178 0.8941368  21.94569 549
## [3] {22697,22698} => {22699} 0.02116178 0.8524845  19.74643 549
```

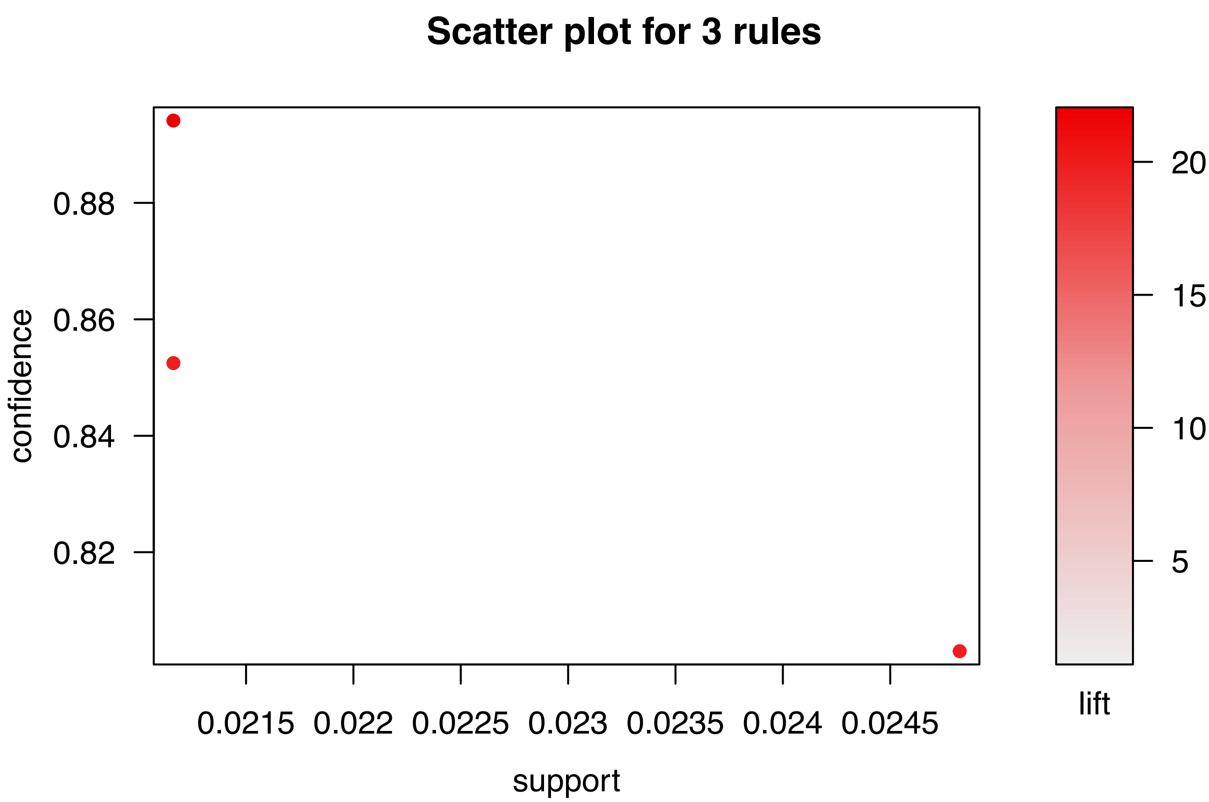
```r
quality(head(rules))
```

```
##      support confidence     lift count
## 1 0.02482365  0.8029925 19.70864   644
## 2 0.02116178  0.8941368 21.94569   549
## 3 0.02116178  0.8524845 19.74643   549
```

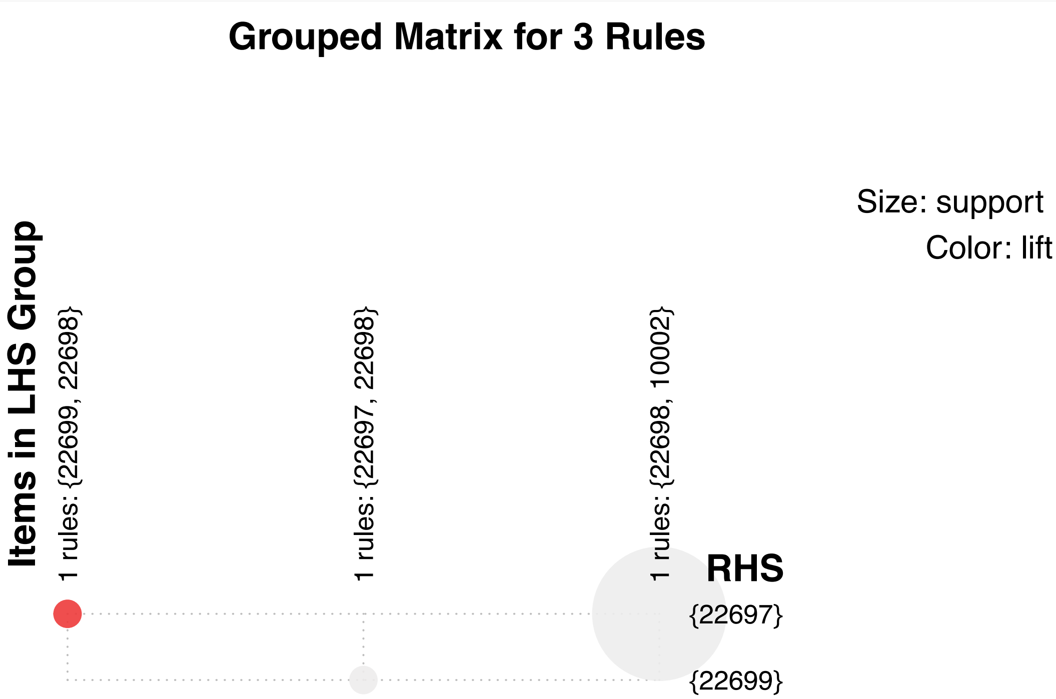
```r
rules <- sort(rules, by="lift")
inspect(head(rules, n=10))
```

```
##     lhs              rhs     support    confidence lift     count
## [1] {22698,22699} => {22697} 0.02116178 0.8941368  21.94569 549
## [2] {22697,22698} => {22699} 0.02116178 0.8524845  19.74643 549
## [3] {22698}       => {22697} 0.02482365 0.8029925  19.70864 644
```

```r
plot(rules)
```




```r
plot(rules, method="grouped matrix")
```




```r
inspect(rules)
```

```
##     lhs              rhs     support    confidence lift     count
## [1] {22698,22699} => {22697} 0.02116178 0.8941368  21.94569 549
## [2] {22697,22698} => {22699} 0.02116178 0.8524845  19.74643 549
```

```
## [3] {22698}       => {22697} 0.02482365 0.8029925  19.70864 644
```

```r
length(rules)
```

```
## [1] 3
```

View 2 - Buying Behavior Itemsets on Transaction Metadata Level

The focus of this view is to explore patterns in transaction metadata, like quantity size with respect to datetime and customer identity. We use our preprocessed data set here with varying values for the support and confidence parameters. Country distorts the itemsets in both view 2 models, regardless of support and confidence. This is due to the retailer being based in the United Kindom with roughly 90% of its customers also based in the United Kingdom. This leads to the apriori rule overvaluing Country in itemsets. While most itemsets contain "United Kingdom", the patterns still exist within the itemsets less the Country variable.

View 2 Model 1 (minlen=2,supp=0.05, conf=0.8)

The parameter selection in this model stems from default values used in the arules documentation. It returns 30 rules due to its lower support value. The most highly supported itemsets in this apriori test were related to the month sold. This highlights the seasonality inherent in these sales data.

```r
  # generate some rules from the frequent itemsets
rules <- apriori(df_pp, parameter = list(minlen=2,supp=0.05, conf=0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5    0.05      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: 27095
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[13058 item(s), 541909 transaction(s)] done [0.45s].
## sorting and recoding items ... [24 item(s)] done [0.02s].
## creating transaction tree ... done [0.33s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [30 rule(s)] done [0.00s].
## creating S4 object  ... done [0.08s].
```

```r
inspect(head(rules, n=10))
```

```
##      lhs                   rhs                      support    confidence
## [1]  {month_sold=4_}  => {Country=United Kingdom} 0.05173931 0.9372242
## [2]  {month_sold=1_}  => {Country=United Kingdom} 0.05886597 0.9076166
## [3]  {month_sold=8_}  => {Country=United Kingdom} 0.05745430 0.8824113
## [4]  {month_sold=3_}  => {Country=United Kingdom} 0.06178713 0.9111516
## [5]  {month_sold=6_}  => {Country=United Kingdom} 0.06121507 0.8996312
## [6]  {month_sold=5_}  => {Country=United Kingdom} 0.06213589 0.9093168
```

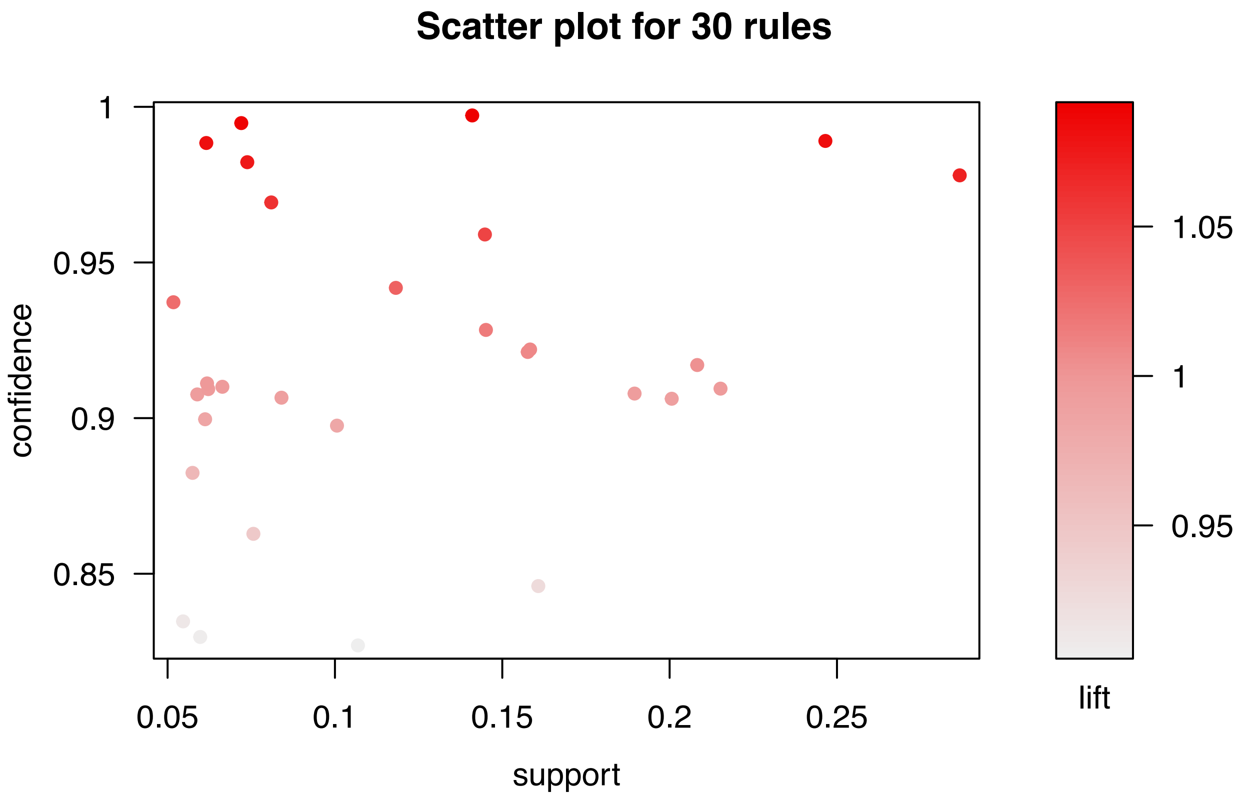
```
## [7]  {month_sold=7_}  => {Country=United Kingdom} 0.06636539 0.9100663
## [8]  {month_sold=9_}  => {Country=United Kingdom} 0.08402333 0.9065623
## [9]  {month_sold=10_} => {Country=United Kingdom} 0.10060914 0.8975832
## [10] {month_sold=12_} => {Country=United Kingdom} 0.11819512 0.9418434
##      lift      count
## [1]  1.0250510 28038
## [2]  0.9926689 31900
## [3]  0.9651016 31135
## [4]  0.9965352 33483
## [5]  0.9839352 33173
## [6]  0.9945284 33672
## [7]  0.9953482 35964
## [8]  0.9915158 45533
## [9]  0.9816953 54521
## [10] 1.0301030 64051
```

```r
quality(head(rules))
```

```
##      support  confidence      lift count
## 1 0.05173931  0.9372242 1.0250510 28038
## 2 0.05886597  0.9076166 0.9926689 31900
## 3 0.05745430  0.8824113 0.9651016 31135
## 4 0.06178713  0.9111516 0.9965352 33483
## 5 0.06121507  0.8996312 0.9839352 33173
## 6 0.06213589  0.9093168 0.9945284 33672
```

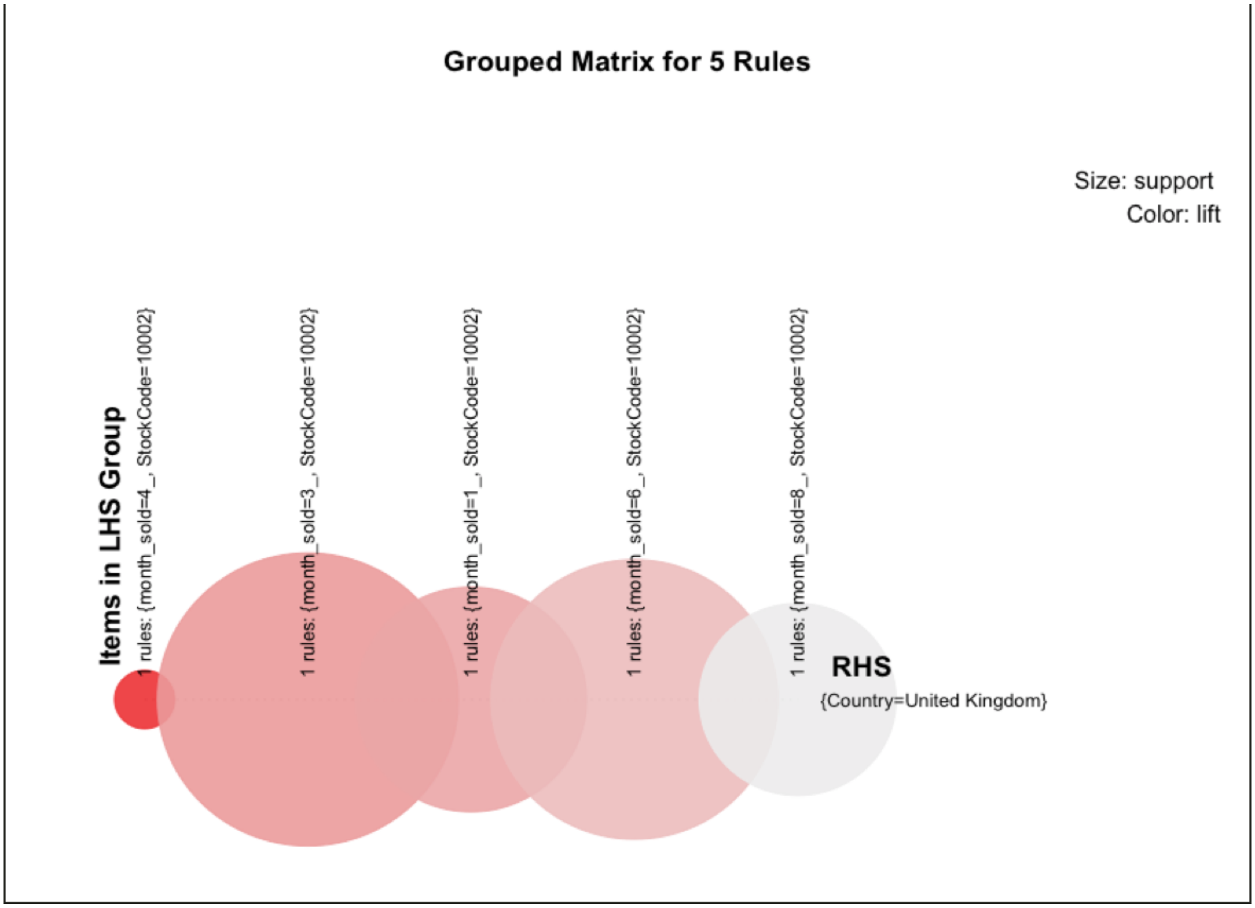
```r
rules <- sort(rules, by="lift")
#inspect(head(rules, n=10))
#interestMeasure(rules[1:10], method=c("phi", "gini"), trans=trans)

plot(rules)
```




```r
# added as img due to rmd scaling issues
# plot(head(rules, n=5), method="grouped matrix")
#inspect(rules)
length(rules)
```

```
## [1] 30
```



View 2 Model 1 (minlen=2,supp=0.1, conf=0.8)

Narrowing the support value yields a cleaner output, with very interesting rankings placed on different months, quantity groups and price groups.

```
  # generate some rules from the frequent itemsets
rules <- apriori(df_pp, parameter = list(minlen=2,supp=0.1, conf=0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      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: 54190
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[13058 item(s), 541909 transaction(s)] done [0.41s].
## sorting and recoding items ... [15 item(s)] done [0.02s].
```

```
## creating transaction tree ... done [0.34s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.10s].
```

```
# Print high support rules
inspect(head(rules, n=10))
```

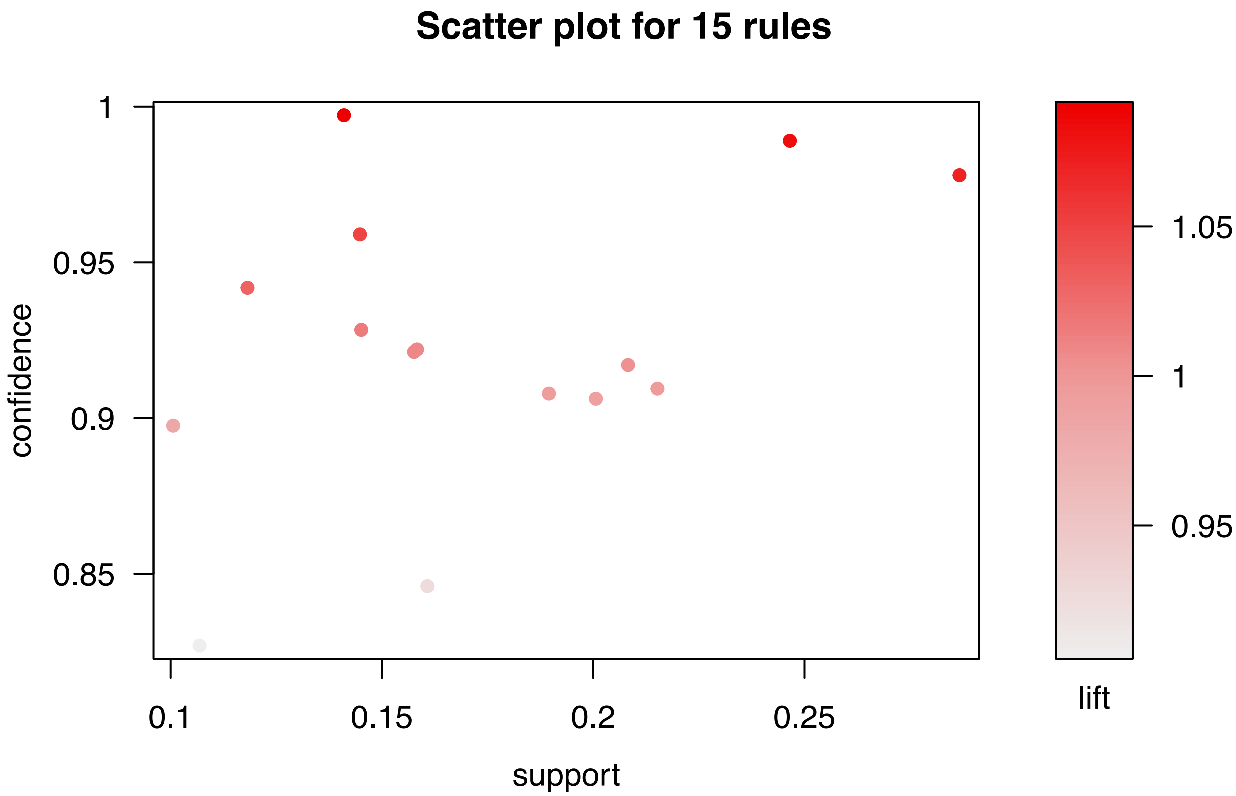
```
##      lhs                           rhs                        support  confidence      lift   count
## [1]  {month_sold=10_}            => {Country=United Kingdom} 0.1006091  0.8975832 0.9816953  54521
## [2]  {month_sold=12_}            => {Country=United Kingdom} 0.1181951  0.9418434 1.0301030  64051
## [3]  {quantity_groups=Qty_Lvl_5} => {Country=United Kingdom} 0.1068648  0.8269810 0.9044770  57911
## [4]  {quantity_groups=Qty_Lvl_2} => {Country=United Kingdom} 0.1448084  0.9589876 1.0488539  78473
## [5]  {month_sold=11_}            => {Country=United Kingdom} 0.1451166  0.9283328 1.0153264  78640
## [6]  {price_groups=Price_Lvl_5}  => {Country=United Kingdom} 0.1576058  0.9212482 1.0075779  85408
## [7]  {price_groups=Price_Lvl_3}  => {Country=United Kingdom} 0.1583089  0.9220460 1.0084504  85789
## [8]  {quantity_groups=Qty_Lvl_4} => {Country=United Kingdom} 0.1607632  0.8460455 0.9253280  87119
## [9]  {price_groups=Price_Lvl_1}  => {Country=United Kingdom} 0.1895226  0.9078887 0.9929665 102704
## [10] {price_groups=Price_Lvl_2}  => {Country=United Kingdom} 0.2006259  0.9062273 0.9911495 108721
```

```
quality(head(rules))
```

```
##     support confidence      lift count
## 1 0.1006091  0.8975832 0.9816953 54521
## 2 0.1181951  0.9418434 1.0301030 64051
## 3 0.1068648  0.8269810 0.9044770 57911
## 4 0.1448084  0.9589876 1.0488539 78473
## 5 0.1451166  0.9283328 1.0153264 78640
## 6 0.1576058  0.9212482 1.0075779 85408
```

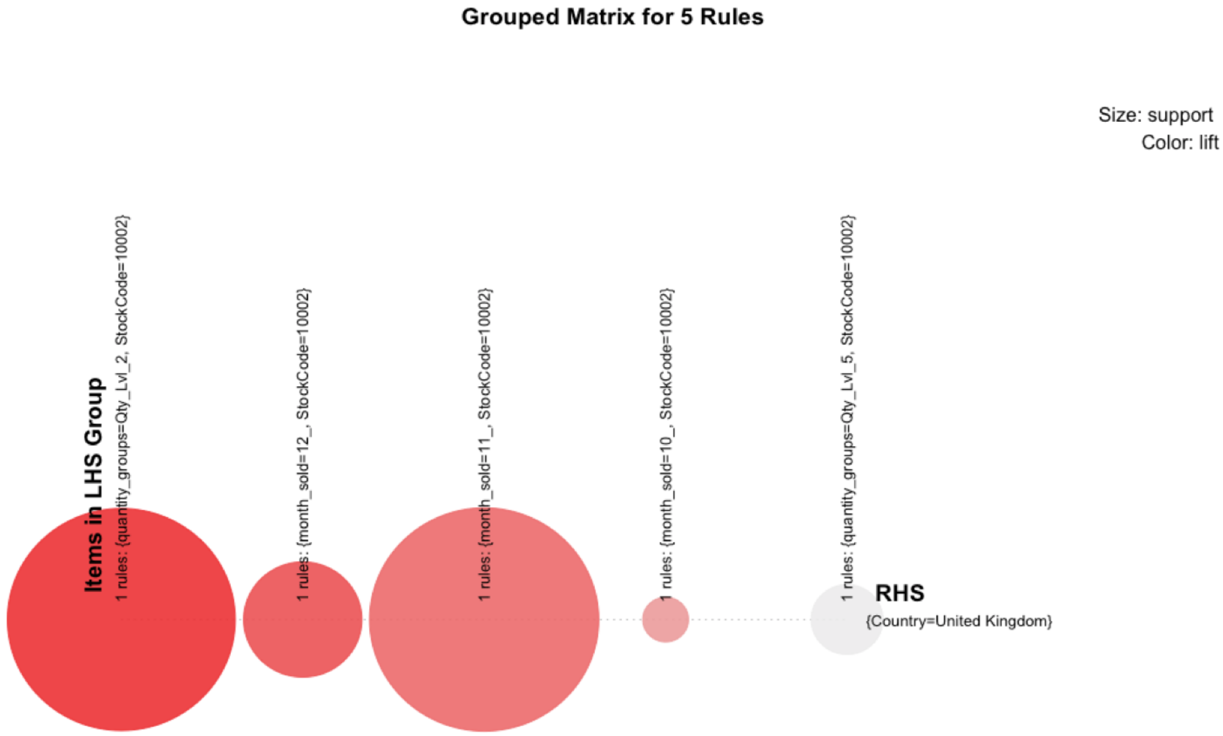
```
rules <- sort(rules, by="lift")
#inspect(head(rules, n=10))
#interestMeasure(rules[1:10], method=c("phi", "gini"), trans=trans)

plot(rules)
```




```r
# added as img due to rmd scaling issues
#plot(head(rules, n=5), method="grouped matrix")
#inspect(rules)
length(rules)
```

```
## [1] 15
```



Deployment

Usage & Further Data Collection

This model serves businesses with information relevant to micro and macro scale trends happening in their industry. The micro scale information is pieced together from our first analysis, looking at what specific clients tend to purchase and how they purchase them. Understanding behaviors like these are key to deploying effective marketing campaigns that can lead to substantial lift. Lift is the increase in purchases from actions taken over what would have occurred if action had not been taken. The macro insights are information regarding the seasonality and volumes of goods purchased at these times. Just-in-time delivery methods and proper inventory management are paramount to running optimized wholesale businesses. This innovation was first made popular when Dell, leveraging a unique statistical understanding of the computer market, changed their delivery method to handle production only at the moment an order was made. Feeding this production method was a carefully implemented inventory system, ensuring that raw materials needed to meet orders always sat at a near zero level, reducing warehousing costs and increasing the average liquidity of business assets.

What types of data would need to be collected to further improve these models? Any purchase data about similar goods would improve the scope, along with general trend data. General trend data allows for the business using these association models to contextualize their internal analyses with the view of the larger market. Context allows for a clear map of where the business sits related to its competitors and insights on how it could improve its own position.

Technical Implementation & Update

The association rules presented above should be implemented into a regular reporting schedule, but not into an automated model. Running similar reviews of the data feeds every quarter would allow for trend tracking and improved insights. This would require an analyst to have access to the data feed and build queries that output current versions of the data we reviewed. Pre-processing scripts that would transform the data into a similar form as used above would allow for an analyst to import the data into R and review the output without much manual calibration.

If a company was inclined to leverage something like this in a product, like an inventory management system, then it should be implemented as one of several signals to purchase new goods. Association rules do not provide enough information alone to be the sole basis of a system.

LSTM Revenue Predictor

1 Exceptional Work - LSTM RNN For Daily Revenue

Fork of https://github.com/llSourcell/How-to-Predict-Stock-Prices-Easily-Demo

- Below we explore the applicability of a Long-Short Term Memory Neural Network for predicting revenue movements on a daily basis. Using Keras, a deep learning library written on top of Google's Tensorflow, we attempt to implement a predictive model.

In [2]:
```
import time
import warnings
import numpy as np
from numpy import newaxis
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import matplotlib.pyplot as plt


warnings.filterwarnings("ignore")

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    print( 'yo')
    #Pad the list of predictions to shift it in the graph to it's correct start
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()

def load_data(filename, seq_len, normalise_window):
    f = open(filename, 'r').read()
    data = f.split('\n')

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
```

```python
    if normalise_window:
        result = normalise_windows(result)

    result = np.array(result)

    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]

def normalise_windows(window_data):
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in window]
        normalised_data.append(normalised_window)
    return normalised_data

def build_model(layers):
    model = Sequential()

    model.add(LSTM(
        input_dim=layers[0],
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
        output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop")
    print( "Compilation Time : ", time.time() - start)
    return model
```

```
def predict_point_by_point(model, data):
    #Predict each timestep given the last sequence of true data,
    # in effect only predicting 1 step ahead each time
    predicted = model.predict(data)
    predicted = np.reshape(predicted, (predicted.size,))
    return predicted

def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run
    # predictions on new window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    #Predict sequence of 50 steps before shifting prediction run forward by 50 steps
    prediction_seqs = []
    for i in range(len(data)//prediction_len):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs
```

```
Using TensorFlow backend.
```

In [3]:
```
import pandas as pd
import datetime as dt
df = pd.read_csv('online_retail.csv')
```

In [4]:
```
df.head()
```

Out[4]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

   InvoiceDate  UnitPrice  CustomerID         Country
```

```
0  12/1/10 8:26       2.55     17850.0  United Kingdom
1  12/1/10 8:26       3.39     17850.0  United Kingdom
2  12/1/10 8:26       2.75     17850.0  United Kingdom
3  12/1/10 8:26       3.39     17850.0  United Kingdom
4  12/1/10 8:26       3.39     17850.0  United Kingdom
```

```
In [5]: df['InvoiceDate'] = pd.to_datetime(df.InvoiceDate)
```

```
In [7]: # run once, intensive process
        df['item_rev'] = df.Quantity * df.UnitPrice
        df = df.drop(['StockCode','Description','Quantity',
                      'UnitPrice','CustomerID','Country'],1)
        df.InvoiceDate = list(map(lambda x: x.replace(hour=0,
                            minute=0,second=0, microsecond=0),
                                  df.InvoiceDate))
```

```
In [8]: data = df.groupby(['InvoiceDate'])['item_rev'].sum()
        data.to_csv('rnn.csv',index=False)
```

```
In [9]: df2 = pd.read_csv('rnn.csv',header=None)
```

```
In [10]: X_train, y_train, X_test, y_test = load_data('rnn.csv', 7, True)
```

```
In [11]: #Step 2 Build Model
         model = Sequential()

         model.add(LSTM(
             input_dim=1,
             output_dim=7,
             return_sequences=True))
         model.add(Dropout(0.2))

         model.add(LSTM(
             100,
             return_sequences=False))
         model.add(Dropout(0.2))

         model.add(Dense(
             output_dim=1))
         model.add(Activation('linear'))

         start = time.time()
         model.compile(loss='mse', optimizer='rmsprop')
         print('compilation time : ', time.time() - start)
```

```
compilation time :  0.026914119720458984
```

```
In [12]: #Step 3 Train the model
         model.fit(
```

```
            X_train,
            y_train,
            batch_size=512,
            nb_epoch=10,
            validation_split=0.05)
```

```
Train on 254 samples, validate on 14 samples
Epoch 1/10
254/254 [==============================] - 1s 5ms/step - loss: 2.3221 - val_loss: 2.2598
Epoch 2/10
254/254 [==============================] - 0s 227us/step - loss: 2.1903 - val_loss: 2.1158
Epoch 3/10
254/254 [==============================] - 0s 208us/step - loss: 2.0906 - val_loss: 1.9788
Epoch 4/10
254/254 [==============================] - 0s 206us/step - loss: 1.9905 - val_loss: 1.8392
Epoch 5/10
254/254 [==============================] - 0s 230us/step - loss: 1.9067 - val_loss: 1.6941
Epoch 6/10
254/254 [==============================] - 0s 234us/step - loss: 1.8167 - val_loss: 1.5479
Epoch 7/10
254/254 [==============================] - 0s 239us/step - loss: 1.7359 - val_loss: 1.4070
Epoch 8/10
254/254 [==============================] - 0s 230us/step - loss: 1.6521 - val_loss: 1.2704
Epoch 9/10
254/254 [==============================] - 0s 227us/step - loss: 1.5523 - val_loss: 1.1470
Epoch 10/10
254/254 [==============================] - 0s 226us/step - loss: 1.4326 - val_loss: 1.0362
```

```
Out[12]: <keras.callbacks.History at 0x1274f7c18>
```

2 Summary of Review

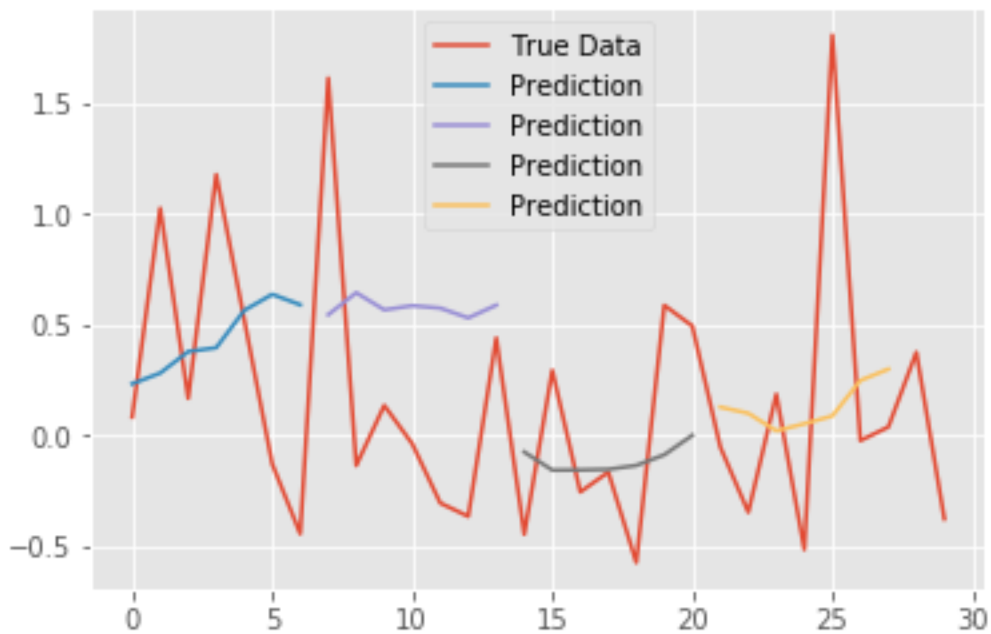
Reviewing the predictions of our LSTM model, we see that the model naturally smoothed the trends, removing most volatility and mapping general macro trends in revenue. Our final model looks at the weekly trends using a range of 7 days for memory. While we expected this window to yield a more representative model, we found that as window size was reduced from an initial size of 50 to 7, volatility of our predicted model increased, but not to the extent that it predicted swings in the short-term.

Our final output maps to the macro movements, which is an interesting result. It highlights both the limitations ands strengths of LSTM methods on small samples. The memory weight vector allows the data to move as important underlying trends change but is unable to identify very short-term movements. As noted in our association rule mining, there are strong seasonal influencers on this purchase data. LSTM does not highlight this on a timescale smaller than annual.

2.1 Predictions

In [13]:
```
#Step 4 - Plot the predictions!
import matplotlib.pyplot as plt
plt.style.use('ggplot')
predictions = predict_sequences_multiple(model, X_test, 7, 7)
plot_results_multiple(predictions, y_test, 7)
```

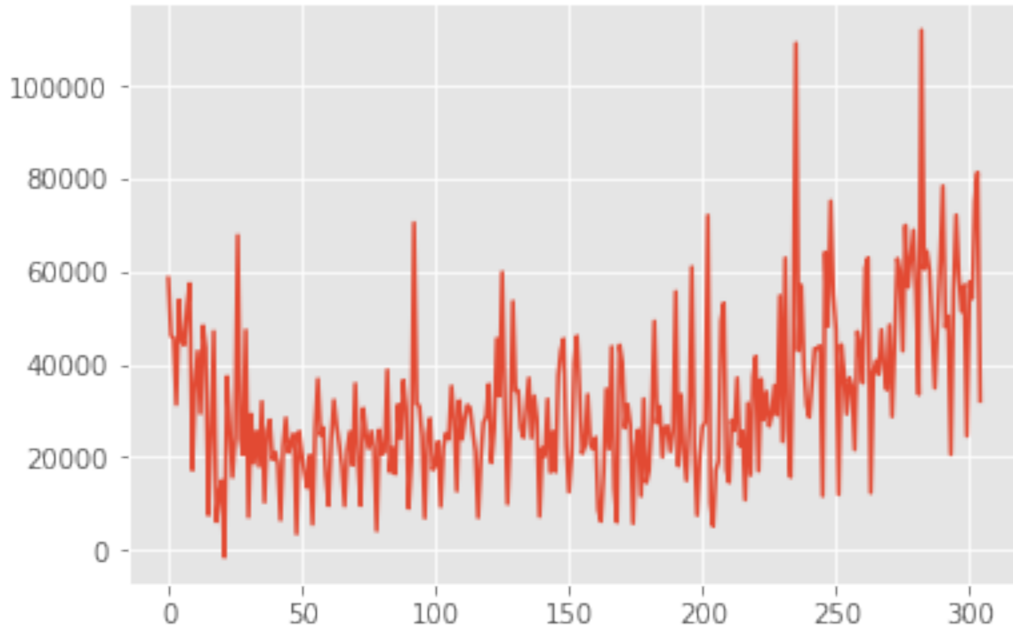
```
yo
```



2.2 Reality

In [14]:
```
%matplotlib inline
plt.plot(df2[0])
```

Out[14]: [<matplotlib.lines.Line2D at 0x1277880f0>]


100000
80000
60000
40000
20000
0
0
50
100
150
200
250
300