Online Retailer Data Analysis

Ying Xie

CIS 133 & CIS 140

The Problem

For the last decade, the retail landscape has changed significantly. Much of the retail is now happening online. I myself for example, buy lots of personal and household items from Amazon. I used to shop at Target often. And nowadays, I barely step my foot into the store.

It seems for a retail business to survive and thrive, the online presence is a must. How can online retail business be more successful? This of course if a huge undertake, and I don't attempt to address it. Instead, for the purpose of this project, I will only try to use the MapReduce, Python and R to address to questions like:

Who are the most "buying" customers?

Also, could I try to make a recommendation like "customers bought this item also bought xxx"?

Why is the problem interesting?

Online retail is a data intensive industry. Therefore, big data analytics can help address many issues the industry is facing. I am sure Amazon has an entire team working on figuring out customer's behaviors and trying to predict the future growth. I am also sure Google is charging retailers handsome fees for displaying targeted s ads after I browse a certain store website.

For the purpose of this project, I want to experience the life cycle of a data analytics project: collecting/gathering the data, cleaning/wrangling the data, and analyzing the data.

The data

Much of the retail data is proprietary. Amazon is not releasing their sales data and customer data, neither does TaoBao (one of the largest online retail platforms in China). It is possible to crawl their websites to get some product related data, but customer data will still be missing.

So, I searched around the internet, and here is a sample dataset from UCI Machine Learning Repository:

http://archive.ics.uci.edu/ml/datasets/online+retail

It is a transactional dataset that contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique alloccasion gifts. Many of its customers are wholesalers (note this is important, as different retailers target different markets)

The beauty of this dataset is to have both product and customer information available.

Data Collection

a) How will you obtained your data?

I will download the data from this url: http://archive.ics.uci.edu/ml/datasets/online+retail

b) How large will your data be?

It's an Excel sheet, with 8 attributes and 541909 instances. The physical size is about 23MB.

c) In what format are your storing your data?

The dataset comes in Excel format, but I will convert it to a CSV file for easier processing.

d) Will you need to process the original data to get it into an easier, more compressed format?

Yes, I will process the original data.

The original dataset has the following attributes: (quoted from UCI website)

- InvoiceNo: Invoice number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.
- StockCode: Product (item) code uniquely assigned to each distinct product.
- Description: Product (item) name.
- Quantity: The quantities of each product (item) per transaction.
- InvoiceDate: Invoice Date and time, the day and time when each transaction was generated.
- UnitPrice: Unit price. Product price per unit in sterling.
- CustomerID: Customer number uniquely assigned to each customer.
- Country: Country name. The name of the country where each customer resides.

1. I notice that in the Quantity column, there are a few rows with negative values. I assume that is some kind of return? Since there are only a few records with negative values, and this project is not investigating customer returns, I will omit these records.

2. One question I want to ask is, who is the most "buying" customer for this retailer?

The dataset only contains UnitPrice and Quantity, so I will need to calculate the PurchasePrice by multiply the UnitPrice with Quantity, and add that column to the dataset.

3. Another question I would like to explore is: At what day and time customers are most likely to place an order? This question is particularly relevant for online retail since the web is on 24*7.

Note: For this project, I didn't explore answers for this question. But I will add these new columns to be used for the next iteration/release.

The original data has the InvoiceDate column, it is in the format of 12/1/2010 8:26:00 AM. I will need to extract the time portion of it, and add as another column to the dataset.

Also, Time is a continuous variable, and I need to put it in a time band. So for the time portion, I will only care about the hour, and omit the minutes and seconds.

I will also add a column called Day to record the Days of the Week.

Please note that I assume that the InvoiceDate column was recorded as local date/time(where the customer was). This would be important for analyzing the customer behaviors.

e) How would you simulate similar data?

As said early, I could crawl the web for some product data. I could also crawl customer review data. But it will be difficult to get customer demographic information and actual purchase history and behaviors, without working for the retailer.

That being said, if I could get my hands onto some proprietary data, then I could possibly explore more buying patterns, such as which kind of customers like to buy which kind of products at what day and time.

<u>Evaluation Report</u>

Steps I have taken for exploring this dataset and get some answers:

1. Download the dataset as Excel sheet.

2. Data Cleaning, Wrangling, and Preparing: To my surprise, there is a lot of work has to be done here.

2.1 First, Add three new columns: PurchasePrice, Day, Time.

It is done in Excel.

For PurchasePrice, use formular: UnitPrice * Quantity

For Day, use function: Text(InvoiceDate, "ddd")

For Time, use function: Hour (to extract only the Hour portion of InvoiceDate)

2.2 Second, Omit the headers. I had to run into multiple "Data Type" errors in order for me to realize that I had to read in the data without the headers.

2.3 Third, Remove the rows with no Customer ID. Again, until I got my hands dirty playing with the data, and running python code on it, I didn't realize that there are rows with missing Customer ID. Since I want to find out "the most buying customers", it would make no sense to keep these rows. So, I deleted them using python with the following code:

```
import pandas as pd
import csv

onlineRetail  = pd.read_csv('OnlineRetail.csv')

onlineRetail _dropna = onlineRetail.dropna()

onlineRetail _dropna.to_csv(onlineRetail _dropna.csv',index=False)
```

2.4 Fourth, Remove the rows with negative quantities. The negative quantities seem to represent returns. And for this project, I am NOT exploring customer returning behaviors, only purchasing behaviors. So I decide to omit these rows as well. The python code is as the following:

```
with open(onlineRetail _dropna.csv', 'r') as inp, open('OnlineRetail_Cleaned.csv', 'w', newline='') as out:

    writer = csv.writer(out)

    for row in csv.reader(inp):

        if int(row[3]) >= 0: //omit the negative quantities

            writer.writerow(row)
```

3. Install Python MapReduce package MRJob

4. Use MapReduce to calculate the most "buying" customer.

I'd like to explore three types of most "buying" customers:

Type 1: The customer bought most quantity

Map Phase:

(CustomerID, Quantity)

Shuffle Phase:

(CustomerID, Quantity, Quantity, Quantity...)

Reduce Phase:

(CustomerID, TotalQuantity)

Here is the corresponding Python code:

```
from mrjob.job import MRJob

class MRCustomerQuantity(MRJob):

    def mapper(self, _, line):

        if not line:

            pass

        else:

             (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice,

            CustomerID, Country, PurchasePrice, Time, DaysOfWeek) = line.split(',')

            yield CustomerID, int(Quantity)

    def reducer(self, CustomerID, Quantity):

        yield CustomerID, sum(Quantity)
```

However, the results are not sorted based on Quantity, but on CustomerID. I want the results to be sorted by Quantity, since that is what I am interested in. Therefore, I modified the python code to do chaining in MapReduce. The idea is, taking the reducer results from the python code above, and sending it to another MapReduce step. In the second step, it will use Quantity as key, CustomerID as value. It is illustrated here:

Reducer (from the first MapReduce step)

Customer 1: 30   Customer 2: 50   Customer 3: 100...

Mapper (now the Quantity is the Key)

30: Customer 1   30: Customer 10   30: Customer 40...

Reducer

1: Customer 15802   1: Customer 15823   1: Customer 16742...

Here is the modified Python code:

```
from mrjob.job import MRJob

from mrjob.step import MRStep

class MRCustomerQuantity(MRJob):

    def steps(self):

        return [

            MRStep(mapper=self.mapper_get_quantity,

                reducer=self.reducer_count_quantity),

            MRStep(mapper=self.mapper_make_counts_key,

                reducer = self.reducer_output_customers)

    ]


   def mapper_get_quantity(self, _, line):

     if not line:

       pass

     else:

       (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice,

       CustomerID, Country, PurchasePrice, Time, DaysOfWeek) = line.split(',')

       yield CustomerID, int(Quantity)


   def reducer_count_quantity(self, CustomerID, Quantity):

     yield CustomerID, sum(Quantity)


  def mapper_make_counts_key(self, CustomerID, count): //second mapper
```

```
      yield '%04d'%int(count), CustomerID


  def reducer_output_customers(self, count, CustomerIDs): //second reducer
      for cid in CustomerIDs:
          yield count, cid
```

<u>The top 10 customers who bought the most quantities are:</u>

Rank	CustomerID	Total Quantity Purchased
1	12830	9848
2	17677	9775
3	17940	9755
4	12921	9526
5	17428	9474
6	12971	9289
7	15039	9209
8	16839	8975
9	16656	8894
10	16133	8888

Type 2: The customer spent the most dollars.

This is very similar to Type 1, except the PurchasePrice would be a float, not an Integer.

Map Phase:

(CustomerID, PurchasePrice)

Shuffle Phase:

(CustomerID, PurchasePrice, PurchasePrice, PurchasePrice...)

Reduce Phase:

(CustomerID, Total PurchasePrice)

```
from mrjob.job import MRJob

from mrjob.step import MRStep


class CustomerPurchase(MRJob):

    def steps(self):

        return [

            MRStep(mapper=self.mapper_get_purchases,

                   reducer=self.reducer_totals_by_customer),

            MRStep(mapper=self.mapper_make_amounts_key,

                   reducer=self.reducer_output_results)

        ]


    def mapper_get_purchases(self, _, line):

         (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice,

         CustomerID, Country, PurchasePrice, Time, DaysOfWeek) = line.split(',')

         yield CustomerID, float(PurchasePrice)




    def reducer_totals_by_customer(self, CustomerID, PurchasePrice):

        yield CustomerID, sum(PurchasePrice)


    def mapper_make_amounts_key(self, CustomerID, purchaseTotal):

        yield '%04.02f'%float(purchaseTotal), CustomerID


    def reducer_output_results(self, purchaseTotal, CustomerIDs):
```

```
for cid in CustomerIDs:
    yield cid, purchaseTotal
```

<u>The top 10 customers who spent the most money:</u> (I am not sure what currency this dataset uses. Let's assume that all records are in the same currency)

Rank	CustomerID	Total Money Spent
1	12722	997.63
2	14540	996.26
3	15110	996.10
4	17594	993.18
5	16009	992.71
6	13363	992.50
7	14223	991.13
8	16232	990.88
9	17324	990.23
10	13703	99.50

Notice something interesting: The total amount of the money spent is very small. The biggest spender is < 1000. Is this result correct? Further inspecting the result, I realized that I made a crucial mistake when trying to converting the float numbers to string for sorting purpose.

Instead of using:

```
yield '%04.02f'%float(purchaseTotal), CustomerID
```

I have to use a string with a much larger length. With a few try and errors, I used the following to get the correct result:

```
yield '%09.02f'%float(purchaseTotal), CustomerID
```

<u>Now the correct result for the top 10 customers who spent the most money</u>:

Rank	CustomerID	Total Money Spent
1	14646	280206.02
2	18102	259657.30
3	17450	194550.79
4	16446	168472.50
5	14911	143825.06
6	12415	124914.53
7	14156	117379.63
8	17511	091062.38

9	16029	081024.84
10	12346	077183.60

Other interesting things noticed:

Customer 13256 had a spending of 0. Furthermore, this customer bought 12540 counts of

"ASSTD DESIGN 3D PAPER STICKERS" in one order but paid 0 for it. Why did the customer pay 0 for this purchase? Now, I do not have more knowledge about this customer or this purchase, but it will be worthwhile to bring it up for further analysis when more data is available.

Type 3: The most frequent buyer

This is a bit tricky. I will count each invoice as one buy, and the customer with the most invoice count will take the prize.

It needs three steps.

First,

The Mapper:

(key: (CustomerID, InvoiceNo), value: 1)

The Reducer:

(key: (CustomerID, InvoiceNo), value: totalCount)

Second,

The Mapper:

(key: [CustomerID, InvoiceNo][0], value: result from the last Reducer)

The Reducer:

(key: [CustomerID, InvoiceNo][0], value: totalCount from the second Mapper)

Third:

The Mapper:

(key: value from the second Reducer, and convert to string, value: key from the second Reducer)

The Reducer:

(key: value from the third Mapper, value: key from the third Mapper)

Here is the Python code:

```
from mrjob.job import MRJob

from mrjob.step import MRStep


class CustomerFrequent(MRJob):

    def steps(self):

        return [

            MRStep(mapper=self.mapper_get_orders,

                   reducer=self.reducer_count_orders),

            MRStep(mapper=self.mapper_get_customers,

                   reducer=self.reducer_count_customers),

            MRStep(mapper=self.mapper_make_counts_key,

                   reducer=self.reducer_output_results)

        ]

    def mapper_get_orders(self, _, line):

        (CustomerID, InvoiceNo) = line.split(',')

        yield ((CustomerID, InvoiceNo), 1)




    def reducer_count_orders(self, key, value):

        yield key, sum(value)


    def mapper_get_customers(self, key, value):
```

```
        yield (key[0], value)


    def reducer_count_customers(self, key, value):
        yield key, sum(value)


    def mapper_make_counts_key(self, key, value):
        yield '%04d'%int(value), key


    def reducer_output_results(self, key, values):
        for v in values:
            yield v, key
```

<u>The top 10 customers who are the most frequent buyers:</u>

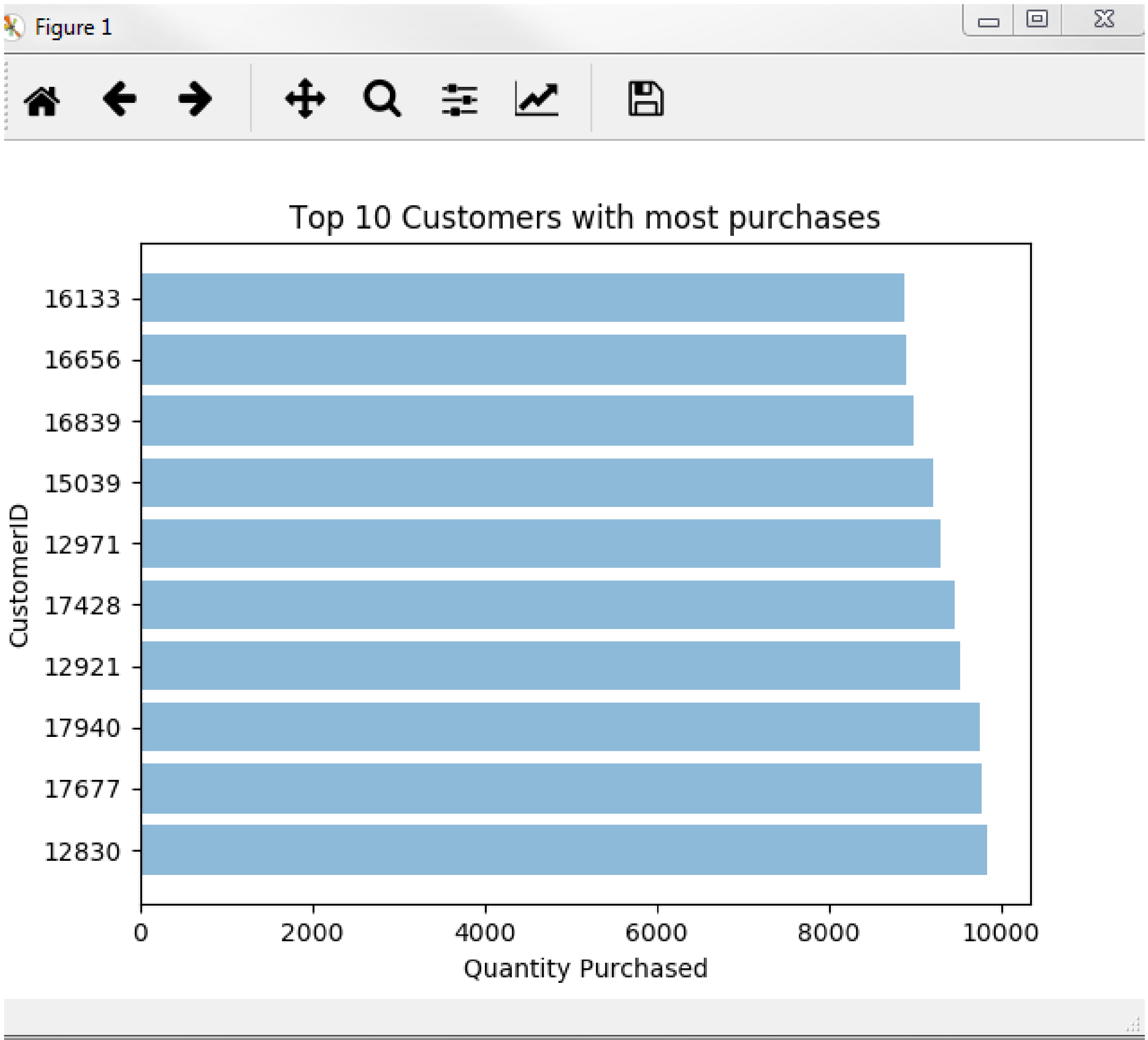
Rank	CustomerID	How Many Buys
1	17841	7847
2	14911	5677
3	14096	5111
4	12748	4596
5	14606	2700
6	15311	2379
7	14646	2080
8	13089	1818
9	13263	1677
10	14298	1637

Lessons Learned:

1. The most important thing I have learned from this project is that data cleaning and prepping is a very important step in data analytics. It may not be sexy or fun, but it is crucial. Without proper data, analytic results can be useless or completely wrong. Be prepared to spend a lot of time on this step.

2. Never trust your results. Always ask questions. Why? Does it make sense? We need domain knowledge, common sense, and a never fading curiosity.

Here are some charts created based on the exploring results.



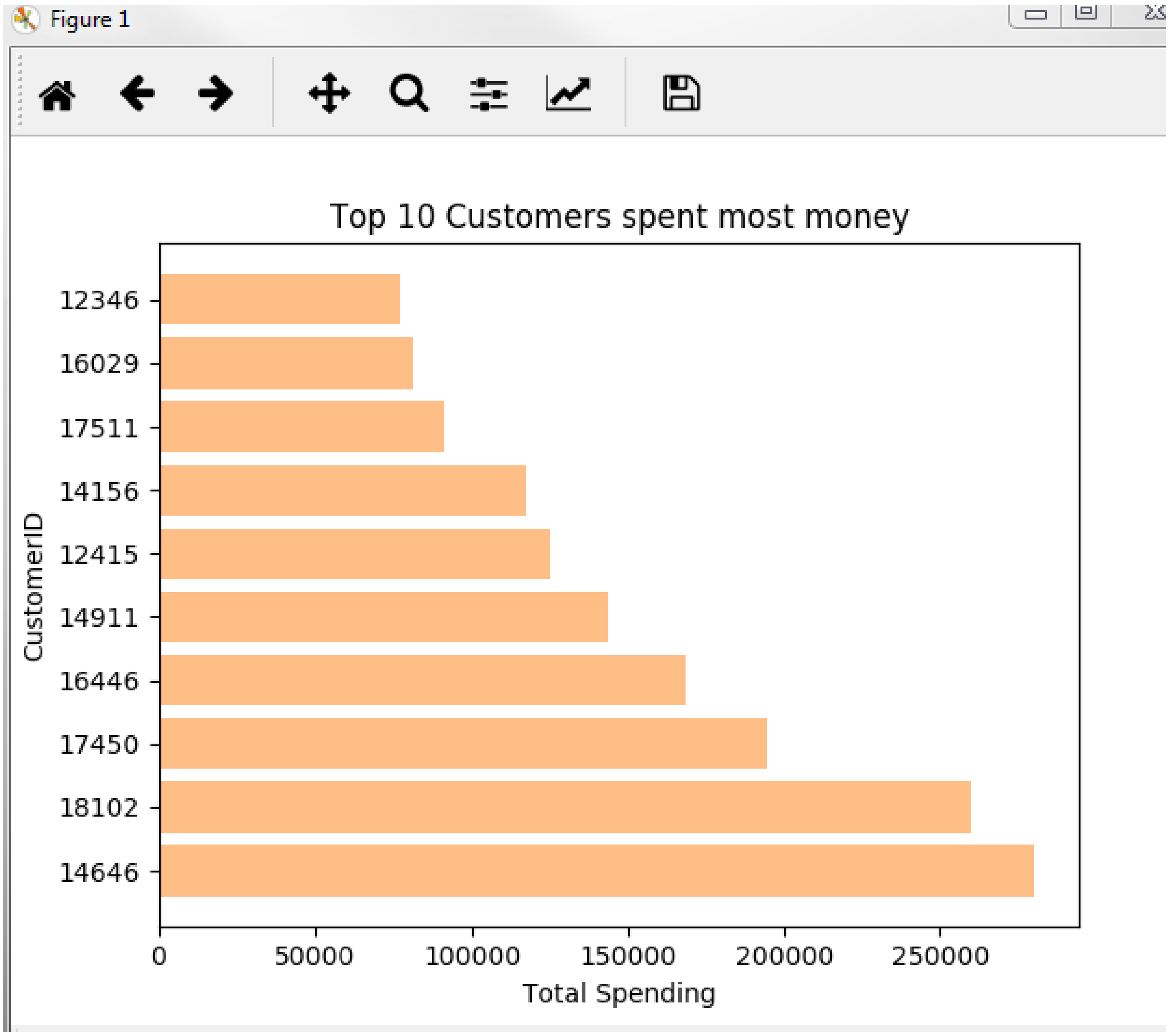


Figure 1
Top 10 Customers spent most money
CustomerID
12346
16029
17511
14156
12415
14911
16446
17450
18102
14646
0
50000
100000
150000
200000
250000
Total Spending

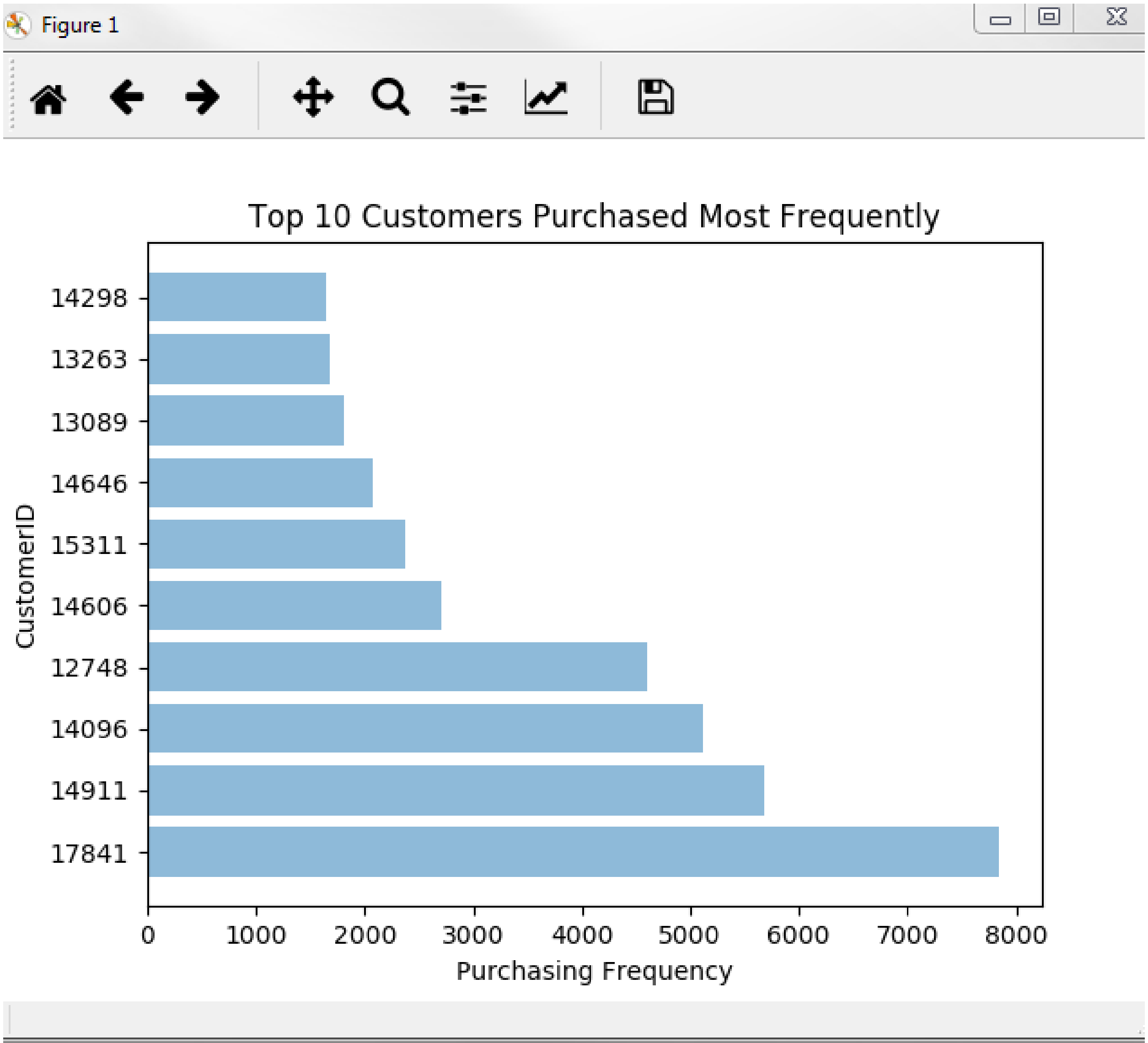


Figure 1
Top 10 Customers Purchased Most Frequently
CustomerID
14298
13263
13089
14646
15311
14606
12748
14096
14911
17841
0
1000
2000
3000
4000
5000
6000
7000
8000
Purchasing Frequency

7. Develop a recommender algorithm using market basket analysis /association rules.

This is done in R.

First and foremost, the dataset needs to be reprocessed.

I have removed the rows without CustomerID, or with negative PurchaseQuantity when doing the analysis for the most buying customers. Now, for this market basket analysis, I will keep the rows without CustomerID, only remove the rows with negative PurchaseQuantity. Because negative quantity probably means returns, and should not be included in this analysis.

This is what the data looks like after removing the negative quantity:

```
'data.frame':   531285 obs. of  11 variables:
 $ InvoiceNo    : Factor w/ 20728 levels "536365","536366",..: 1 1 1 1 1 1 1 2 2 3 ...
 $ StockCode    : Factor w/ 3941 levels "10002","10080",..: 3426 2739 2966 2909 2908 1628 775
...
 $ Description  : Factor w/ 4078 levels ""," 4 PURPLE FLOCK DINNER CANDLES",..: 3892 3900 905
1512 1637 1634 242 ...
 $ Quantity     : int  6 6 8 6 6 2 6 6 6 32 ...
 $ InvoiceDate  : Factor w/ 19052 levels "1/10/2011 10:32",..: 5698 5698 5698 5698 5698 5698 5
700 ...
 $ UnitPrice    : num  2.55 3.39 2.75 3.39 3.39 7.65 4.25 1.85 1.85 1.69 ...
 $ CustomerID   : int  17850 17850 17850 17850 17850 17850 17850 17850 17850 13047 ...
 $ Country      : Factor w/ 38 levels "Australia","Austria",..: 36 36 36 36 36 36 36 36 36 36
 $ PurchasePrice: num  15.3 20.3 22 20.3 20.3 ...
 $ Time         : int  8 8 8 8 8 8 8 8 8 8 ...
 $ DaysofWeek   : Factor w/ 6 levels "Friday","Monday",..: 6 6 6 6 6 6 6 6 6 6 ...
```

Then, for market analysis using association rules, I only need products in each order. Each row represents one order, and each column represents one product purchased within that order. The dataset should look like the following:

	A	B	C	D	E
1	**Product1**	**Product2**	**Product3**	**Product4**	**Product5**
2	WHITE HANGING HEART T-LIGHT HO	WHITE METAL LANTERN	CREAM CUPID HEARTS COAT HA	KNITTED UNION FLAG HOT WATER BC	RED WOOLLY HOTTIE
3	HAND WARMER UNION JACK	HAND WARMER RED POLKA DOT			
4	ASSORTED COLOUR BIRD ORNAME	POPPY'S PLAYHOUSE BEDR	POPPY'S PLAYHOUSE KITCHEN	FELTCRAFT PRINCESS CHARLOTTE I	IVORY KNITTED MUG
5	JAM MAKING SET WITH JARS	RED COAT RACK PARIS FAS	YELLOW COAT RACK PARIS FASH	BLUE COAT RACK PARIS FASHION	
6	BATH BUILDING BLOCK WORD				
7	ALARM CLOCK BAKELIKE PINK	ALARM CLOCK BAKELIKE RE	ALARM CLOCK BAKELIKE GREEN	PANDA AND BUNNIES STICKER SHEE	STARS GIFT TAPE
8	PAPER CHAIN KIT 50'S CHRISTMAS				
9	HAND WARMER RED POLKA DOT	HAND WARMER UNION JACK			
10	WHITE HANGING HEART T-LIGHT HO	WHITE METAL LANTERN	CREAM CUPID HEARTS COAT HA	EDWARDIAN PARASOL RED	RETRO COFFEE MUG

Using the following steps to transform the original dataset, and write the new dataset into a CSV file:

```
> retail <- read.csv(file="C:/Users/Ying/market_basket_no_negative.csv", header=TRUE, sep=",")
> productList <- ddply(retail, c("CustomerID", "InvoiceDate"), function(df1)paste(df1$Description, collapse="
,"))
> productList$CustomerID <- NULL
> productList$InvoiceDate <- NULL
```

```
> write.csv(productList, "market_basket_analysis.csv", quote=FALSE)
```

Now, the dataset looks like this:

MEDIUM CERAMIC TOP STORAGE JAR														
PINK NEW	BLUE NEW	BLACK CA	WOODLAI	AIRLINE B.	AIRLINE B.	SANDWIC	ALARM CL	ALARM CL	ALARM CL	ALARM CL	ALARM CL	SMALL HE.	72 SWEET	60 TEAT
MINI LIGH	PINK GOO	MADRAS I	AIRLINE B.	AIRLINE B.	AIRLINE B.	AIRLINE B.	BIRDCAGE	CHRISTMA	REGENCY (	REGENCY T	TEA TIME I	TEA TIME I	TEA TIME (	PINK RE
BLACK CA	AIRLINE B.	COLOUR G	MINI PAIN	CLEAR DR.	PINK DRA	GREEN DR	RED DRAW	PURPLE DI	BLUE DRA	ALARM CL	ALARM CL	ALARM CL	ALARM CL	ALARM
CLASSIC C	BICYCLE PI	BOOM BO	PINK NEW	RED TOAD	RABBIT NI	WOODLAI	PINK GOO	CHRISTMA	MINI PLAY	MINI PLAYING CARDS DOLLY GIRL				
AIRLINE B.	AIRLINE B.	AIRLINE B.	AIRLINE B.	RED RETR(	ICE CREAM	VINTAGE I	HOLIDAY F	TREASURE	WATERIN(	RED DRAW	LARGE HE.	SMALL HE.	PACK OF 6	RED RET
RABBIT NI	REGENCY T	REGENCY T	REGENCY T	REGENCY T	REGENCY T	REGENCY S	REGENCY I	AIRLINE B.	AIRLINE B.	VICTORIA	NAMASTE	TRIPLE HO	SMALL HE.	3D DOG
SET OF 60	SET 40 HE.	AIRLINE B.	AIRLINE B.	AIRLINE B.	AIRLINE B.	AIRLINE B.	WOODLAI	WOODLAI	ALARM CL	TRIPLE HO	SINGLE AN	TEA TIME (	72 SWEET	60 TEAT
PACK OF 1	PACK OF 1	MULTI HE.	PACK OF 1	PACK OF 1	POSTAGE									
72 SWEET	60 CAKE C.	60 TEATIM	60 TEATIM	PACK OF 7	PACK OF 7	PACK OF 1	PACK OF 1	PACK OF 1	SWEETIES	SET OF 72	SET OF 72	60 CAKE C.	60 CAKE C.	PACK O

From this new dataset, let's derive the transactions for the association rules.

```
> trans <- read.transactions('C:/Users/Ying/market_basket_analysis.csv', format='basket', sep=',')
There were 50 or more warnings (use warnings() to see the first 50)
> summary(trans)

transactions as itemMatrix in sparse format with
 10613 rows (elements/itemsets/transactions) and
 18099 columns (items) and a density of 0.00146601

most frequent items:
WHITE HANGING HEART T-LIGHT HOLDER           REGENCY CAKESTAND 3 TIER             JUMBO BAG RED RETROSPOT
                              1305                               1166                                1003
                     PARTY BUNTING      ASSORTED COLOUR BIRD ORNAMENT                             (Other)
                               968                                931                              276225

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21
   1  523  344  338  339  367  345  349  331  321  312  321  297  264  317  291  270  268  233  269  241
  22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40   41   42
 215  191  173  167  154  147  118  126  147  148  119   94   89   93   91   87   77   63   80   65   62
  43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63
  60   75   45   41   64   52   37   44   36   56   41   40   45   46   25   33   34   30   20   31   27
  64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80   81   82   83   84
  22   21   16   37   36   25   17   31   21   14   19   15   21   21   14   15   12   19   15   14   12
  85   86   87   88   89   90   91   92   93   94   95   96   97   98   99  100  101  102  103  104  105
  14   13   14    9   11    6    9    6    8    8   11   11    9    6   14    5   10    5    5    8    4
```

Now I can create some association rules using Apriori algorithm. R has a built in package for this algorithm.

library(arules)

library(arulesViz)

At first, I set support at 0.6 and confidence at 0.8. And it returned 0 set of rules. Then I tried support at 0.1 and confidence at 0.8. And it still returned 0 set of rules. I continued to decrease the support level in order to produce association rules.

```
> rules <- apriori(tr, parameter = list(supp=0.01, conf=0.8))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5    0.01      1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 106

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[18099 item(s), 10613 transaction(s)] done [0.03s].
sorting and recoding items ... [766 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.02s].
writing ... [249 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> rules <- sort(rules, by='confidence', decreasing = TRUE)
> summary(rules)
set of 249 rules

rule length distribution (lhs + rhs):sizes
  2   3   4   5
 47 130  64   8
```

I only want to see the top 10 rules:

```
> inspect(rules[1:10])
     lhs                           rhs                   support    confidence lift     count
[1]  {FRONT  DOOR}              => {KEY FOB}             0.01008197 1          42.96761 107
[2]  {BACK DOOR}                => {KEY FOB}             0.01460473 1          42.96761 155
[3]  {SUGAR}                    => {SET 3 RETROSPOT TEA} 0.01639499 1          60.99425 174
[4]  {SET 3 RETROSPOT TEA}      => {SUGAR}               0.01639499 1          60.99425 174
[5]  {SUGAR}                    => {COFFEE}              0.01639499 1          46.96018 174
[6]  {SET 3 RETROSPOT TEA}      => {COFFEE}              0.01639499 1          46.96018 174
[7]  {SHED}                     => {KEY FOB}             0.01696033 1          42.96761 180
[8]  {BACK DOOR,SHED}           => {KEY FOB}             0.01055310 1          42.96761 112
[9]  {SET 3 RETROSPOT TEA,SUGAR} => {COFFEE}             0.01639499 1          46.96018 174
[10] {COFFEE,SUGAR}             => {SET 3 RETROSPOT TEA} 0.01639499 1          60.99425 174
>
```

The problem with this result is: There is nothing new here to recommend. Of course people who bought "Front Door" would also buy "Key Fob", or people who bought "Tea" would also buy "Coffee" or "Sugar". This result sees useless. Can I do better?

I decided to decrease support level even more, to 0.001.

```
> inspect(rules[1:10])
     lhs                                   rhs                                   support confidence     lif
t count
[1]  {STICKY GORDON}                    => {GREETING CARD}                    0.001130689          1 221.104
2    12
[2]  {OVERCROWDED POOL.}                => {GREETING CARD}                    0.001036465          1 221.104
2    11
[3]  {YELLOW/PINK FLOWER DESIGN BIG MUG} => {PINK/GREEN FLOWER DESIGN BIG MUG} 0.001036465          1 758.071
4    11
[4]  {NEW ENGLAND}                      => {TUMBLER}                          0.001036465          1 758.071
4    11
[5]  {TREES}                            => {CHRISTMAS GARLAND STARS}          0.001507585          1 663.312
5    16
[6]  {CHRISTMAS GARLAND STARS}          => {TREES}                            0.001507585          1 663.312
5    16
[7]  {DOUGHNUTS}                        => {SQUARE}                           0.001130689          1 884.416
7    12
[8]  {SQUARE}                           => {DOUGHNUTS}                        0.001130689          1 884.416
7    12
[9]  {DOUGHNUTS}                        => {GREETING CARD}                    0.001130689          1 221.104
2    12
[10] {SQUARE}                           => {GREETING CARD}                    0.001130689          1 221.104
2    12
```

Now the rules start to have some meanings. Like "Overcrowded Pool" with "Greeting Card", or "Doughnuts" with "Square" or "Greeting Card".

I want explore more rules. How about the next 10 rules?

```
> inspect(rules[10:20])
     lhs                          rhs                    support     confidence lift     count
[1]  {SQUARE}                  => {GREETING CARD}     0.001130689 1          221.1042 12
[2]  {PINK  SPOTS}             => {SWISS ROLL TOWEL}  0.001413361 1          225.8085 15
[3]  {WOBBLY CHICKEN}          => {DECORATION}        0.002261378 1          246.8140 24
[4]  {WOBBLY CHICKEN}          => {METAL}             0.002261378 1          246.8140 24
[5]  {WOBBLY RABBIT}           => {DECORATION}        0.002732498 1          246.8140 29
[6]  {WOBBLY RABBIT}           => {METAL}             0.002732498 1          246.8140 29
[7]  {DECOUPAGE}               => {GREETING CARD}     0.001790257 1          221.1042 19
[8]  {FUNK MONKEY}             => {ART LIGHTS}        0.002920946 1          342.3548 31
[9]  {ART LIGHTS}              => {FUNK MONKEY}       0.002920946 1          342.3548 31
[10] {BILLBOARD FONTS DESIGN}  => {WRAP}              0.002544050 1          365.9655 27
[11] {NURSERY A}               => {C PAINTED LETTERS} 0.004616979 1          216.5918 49
```

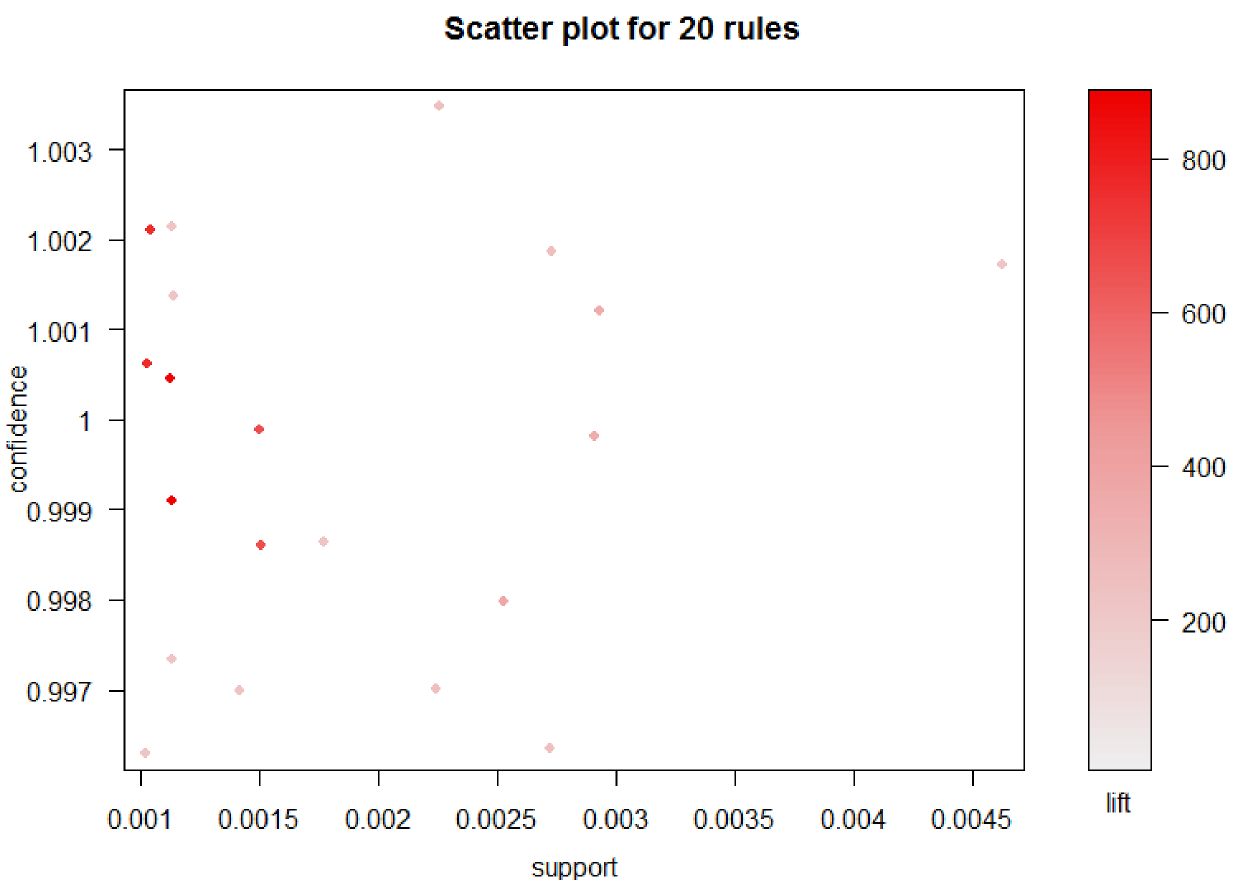
Now the results are getting interesting. "Wobbly Chicken or Rabbit" seems like a popular "Decoration" idea, and "Funk Monkey" is in love with "Art Lights", etc.

A few things noticed about this online retailer according to this dataset:

1. This obviously is a small retailer. It does NOT have the sheet amount of big data like Amazon to mine the rules.

2. The support level is really low. At 0.001, it means these products were only bought at 0.1% of the time (or more). However, the confidence level of 1 means 100% of customers who bought the item in the "lhs" column would also buy the item in column "rhs". Unlike bricks and mortar retailers, it does NOT increase much of the cost for an online retailer to recommend the items even if there is only 0.1% of time these items are being bought. Small online retailers do not have the resources like Amazon to develop sophisticated recommendation system, but it can still use simple algorithms like this one to improve the customer service and possibly sales.

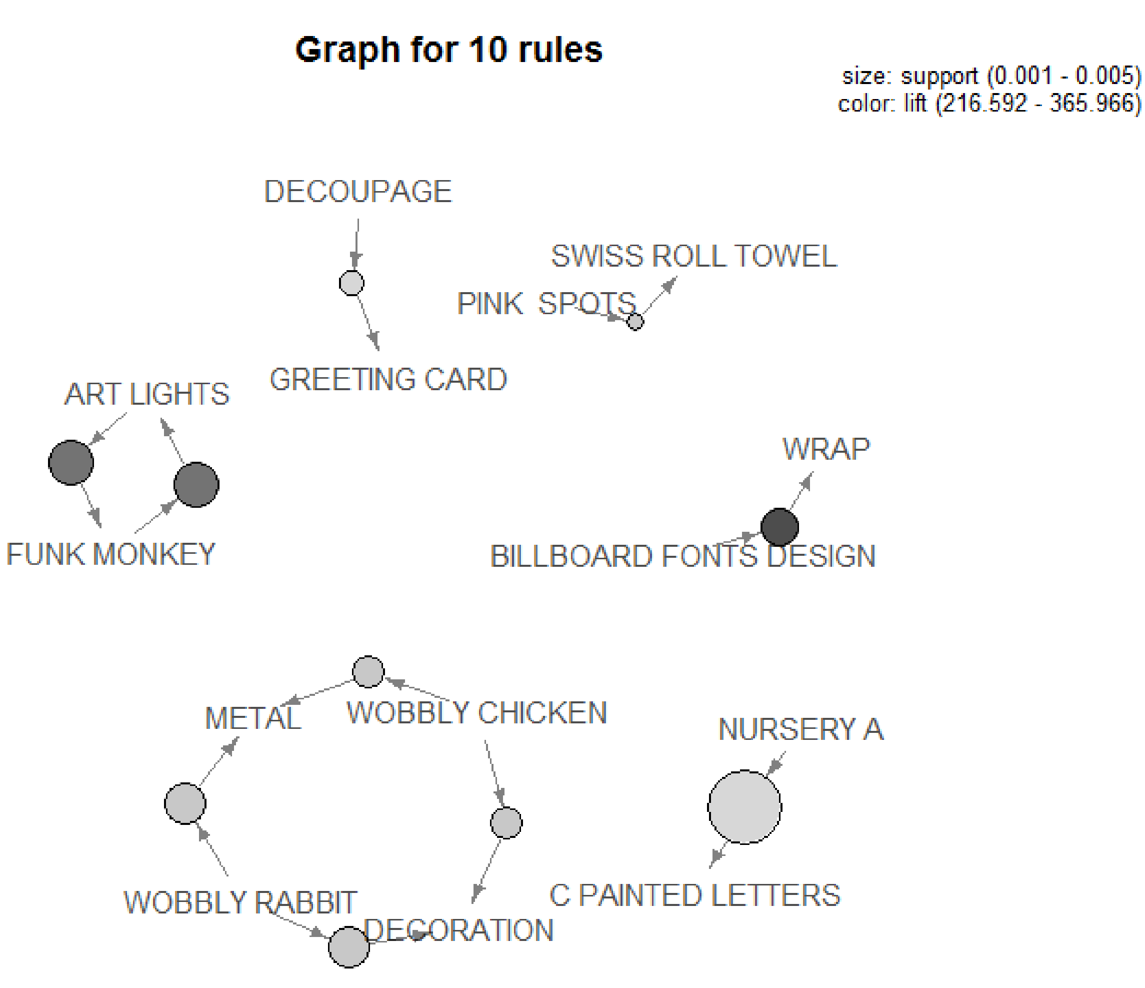
Let's plot these 20 rules:



Plot a graph representing the rules. Since graph does not work well with too many data points, I am using the rules[11:20] to plot the graph. It seems rules[11:20] are more interesting.

```
> plot(top11To20, method="graph",
+       nodeCol = grey.colors(8), edgeCol = grey(.5), alpha = 1)
>
```

The larger circles imply higher support, and the darker color imply higher lift.



Summary: This is an interesting project. I use Excel, MapReduce, Python and R to clean, wrangle, manipulate, analyze, visualize the data, and to make a recommender system. There is much more can be done, and this is a good start.