Overall Status:

As specified in the project description we first cleaned the data in the preprocessing step. After that we assigned integer value to the item description and attached that data to the respective invoice number. This data was converted into basket format and then that was converted into transactions. Using apriori algorithm we generated frequent item set (using apriori function) and candidate item set (using user defined function). Then we generated association rules using all the 9 combinations of support and confidence with the transactions. After that we filtered out 10 rules each for lift>10 and lift<10. Out of all the 9 combinations we visualized top 100 in descending order of minimum confidence.

File Description:

fully_processed_with_whole_data_required_columns_retail_data.csv

This file contains data after removing all the vales that were asked to drop in the project description.

integer_with_description_required_columns_retail_data.csv

Storing integer value with item description

after_integer_required_columns_retail_data.csv

This file contains the integer value of the item description and it is attached to respective associated invoice number.

market_basket_transactions.csv

Data is being stored in basket format building the transactions.

Division of Labor:

Nirmit Shah – Preprocessing, Frequent item set and candidate item set

Mit Patel – Apriori rule generation and visualization

Problems encountered and solution:

Creating own function to assign integer value to each item description was a difficult code to build but we managed to do it. There is no function to create candidate item set so we build our own function which performs brute force to generate candidate item set.

1. Preprocessing:

- Remove the unwanted columns from the given data set
- Removed the records with invoice numbers starting with ‘c’

```
required_columns_retail_data<-required_columns_retail_data[!grepl('^[C-c]', required_columns_retail_data$InvoiceNo),]
```

- Discarding various words that denote actions that is they are NOT items bought

```
required_columns_retail_data <-required_columns_retail_data[!grepl("WRONG", required_columns_retail_data$Description),]
```

- Selecting each item description from the data set and storing it in a form of (key, value) pair and then attaching that key to the corresponding invoice number of the item description.

```
varx <- 1
h<-hash()
demo <-function(a1)
{
  a1<-tolower(a1)
  a1 <- str_squish(a1)
  length_of_h <- length(h)
  if(length_of_h == 0)
  {

    .set(h,keys=varx,values=a1)
     varx<<-varx+1
    return (varx-1)
  }
  else
  {
    for (i in 1:length_of_h)
    {
      j <- toString(i)
      if(values(h,keys=j) == a1)
      {
        return(j)
      }
    }
      .set(h,keys=varx,values=a1)
      varx<<-varx+1

      return(varx-1)
    }


}
trial_dataset<-required_columns_retail_data
trial_dataset <- na.omit(trial_dataset)
len<-dim(trial_dataset)
finallen<-as.numeric(len[1])
for (i in 1:finallen)
{
  trial_dataset$Description[i]<-demo(trial_dataset$Description[i])
}
write.csv(trial_dataset,'after_integer_required_columns_retail_data.csv', row.names = FALSE)
```

Figure 1.1: Block of code which generated the key value pair

	A	B
1	InvoiceNo	Description
2	536365	1
3	536365	2
4	536365	3
5	536365	4
6	536365	5
7	536365	6
8	536365	7
9	536366	8
10	536366	9
11	536367	10
12	536367	11
13	536367	12
14	536367	13

	A	B	C	D
1	NO	Description		
2	1	white hanging heart t-light holder		
3	2	white metal lantern		
4	3	cream cupid hearts coat hanger		
5	4	knitted union flag hot water bottle		
6	5	red woolly hottie white heart.		
7	6	set 7 babushka nesting boxes		
8	7	glass star frosted t-light holder		
9	8	hand warmer union jack		
10	9	hand warmer red polka dot		
11	10	assorted colour bird ornament		
12	11	poppy's playhouse bedroom		
13	12	poppy's playhouse kitchen		
14	13	feltcraft princess charlotte doll		
15	14	ivory knitted mug cosy		
16	15	box of 6 assorted colour teaspoons		

Figure 1.1.1: Invoice number with respective item description key Figure 1.1.2: Key attached to item description

- Converting the data into Txs

```
transaction_data<-read.csv("after_integer_required_columns_retail_data.csv", sep = ",", header = TRUE)
retail<-read.csv("after_integer_required_columns_retail_data.csv",sep=",")
transactionData <- ddply(retail,c("InvoiceNo"),
                         function(df1)paste(df1$Description,
                                            collapse = ","))
transactionData$InvoiceNo <- NULL
write.csv(transactionData,"market_basket_transactions.csv", quote = FALSE, row.names = FALSE)
tr <- read.transactions('market_basket_transactions.csv', format = 'basket', sep=',')
```

Figure 1.2: This block of code will convert the data into basket format transaction

```
market_basket_transactions - Notepad
File  Edit  Format  View  Help
items
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
27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46
47
9,8
1,2,3,48,49,50,51,52,53,54,55,4,5,6,7
56
1,2,3,48,49,50,51,52,53,54,55,4,5,6,7
57,58
9,8
59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77
```

Figure 1.2.1: Output of basket format

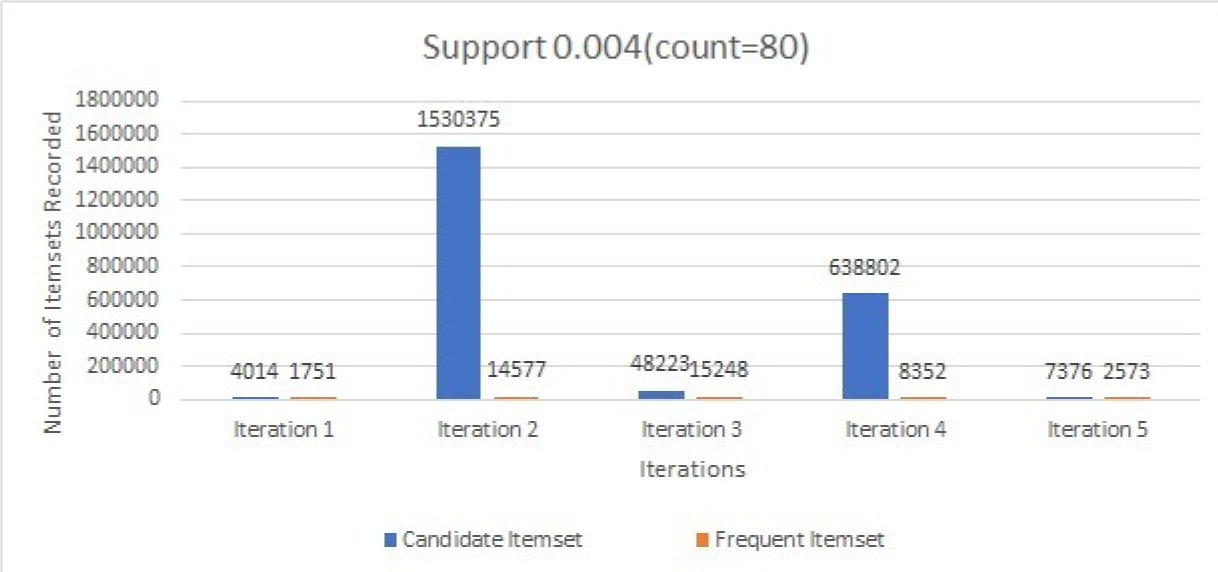
<u>2. Analysis of candidate item set *and frequent items set*:</u>

```
retrive<- as.data.frame(frequent_itemset_004_l1@items@data@i)
names(retrive)[1]<-"items"
retrive1<- as.data.frame(frequent_itemset_004_l1@items@data@i)
names(retrive)[1]<-"items1"
pubsperauthor <- crossing(retrive$items1,retrive1$`frequent_itemset_004_l1@items@data@i`)
names(pubsperauthor)[2]<-"items1"
names(pubsperauthor)[1]<-"items"
calculate_candidate_2_2007 <- filter(pubsperauthor,  items< items1 &items!=0 & items1!=0 )
data_combine<-paste(pubsperauthor$items,',',pubsperauthor$items1)
write.csv(data_combine,"checker.csv", quote = FALSE, row.names = FALSE)
```

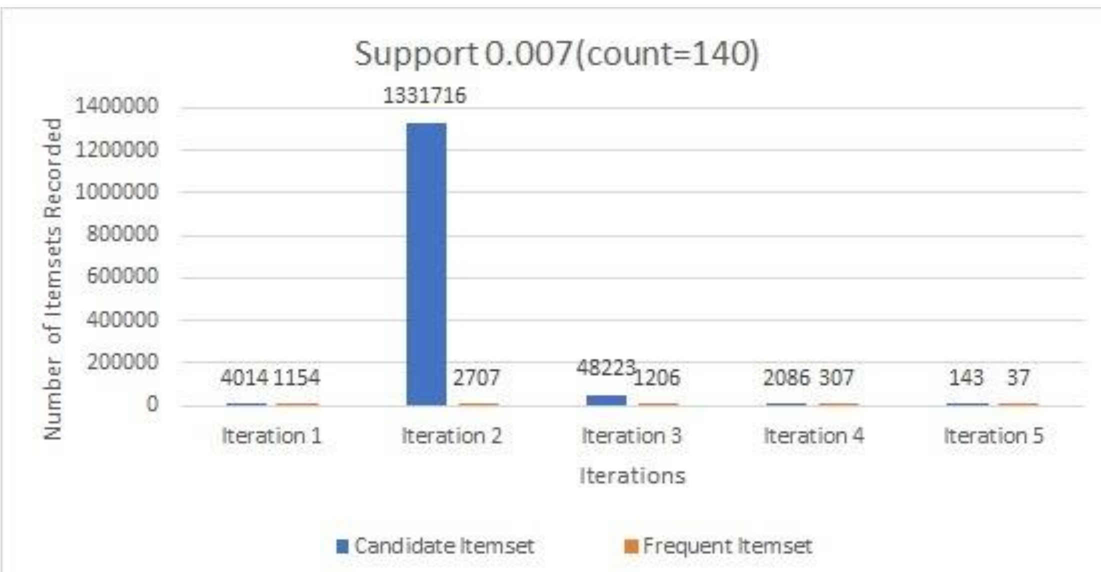
```
frequent_itemset_004_2<- eclat(data = tr, parameter = list(support =0.004, maxlen = 2,minlen = 2))
#4. C3
dfft <-  select(inspect(frequent_itemset_004_2), items)
for (i in 1:nrow(dfft))
{
  dfft[i,]<-str_remove(dfft[i,], "[{ }]")
  dfft[i,]<-str_remove(dfft[i,], "[ }]")
}
write.csv(dfft,"c2.csv", quote = FALSE, row.names = FALSE)
fdfft<-read.csv("c2.csv",sep=",")
fdfft1<-fdfft
names(fdfft1)[2]<-"items4"
names(fdfft1)[1]<-"items3"
names(fdfft)[2]<-"items2"
names(fdfft)[1]<-"items1"
demo_trial<- crossing(fdfft,fdfft1)
calculate_candidate_itemsets_3 <- filter(demo_trial,  items1==items3 &items2<items4)
calculate_candidate_itemsets_3<- subset(calculate_candidate_itemsets_3, select = -c(items3))
data_combine1<-paste(calculate_candidate_itemsets_3$items1,',',calculate_candidate_itemsets_3$items2,',',calculate_candidate_itemsets_3$items4)
write.csv(data_combine1,"checker1.csv", quote = FALSE, row.names = FALSE)
```

Storing the result of frequent item set from iteration 1 and performing self-join on it along with we also check the condition specified in algorithm of generating candidates using which frequent item set for iteration 2 will be generated. We used different use defined function to generate each candidate item set iteration which will replicate apriori generate candidate algorithm.

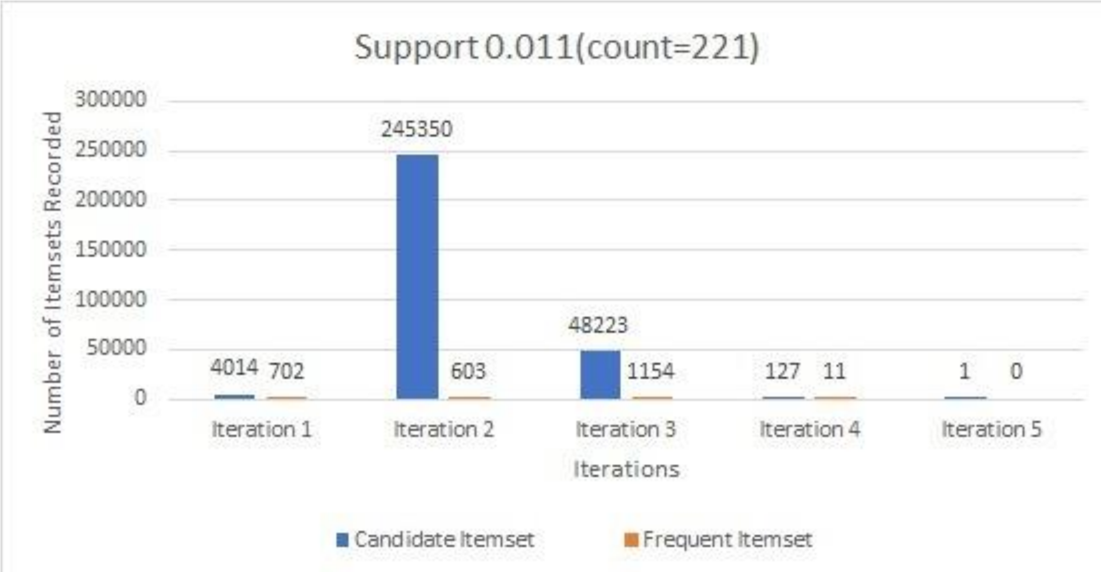
From all the graphs shown below we can conclude that the number of itemsets recorded in candidate itemset is always greater than number of itemsets recorded in frequent itemsets because we used self-join in our user defined function.



When the support is 0.004 (count = 80) we got 4014 items in candidate itemset and 1751 in frequent itemset after first iteration which changed to 7376 for candidate itemset and 2573 for frequent itemset by the end of the iteration 5.

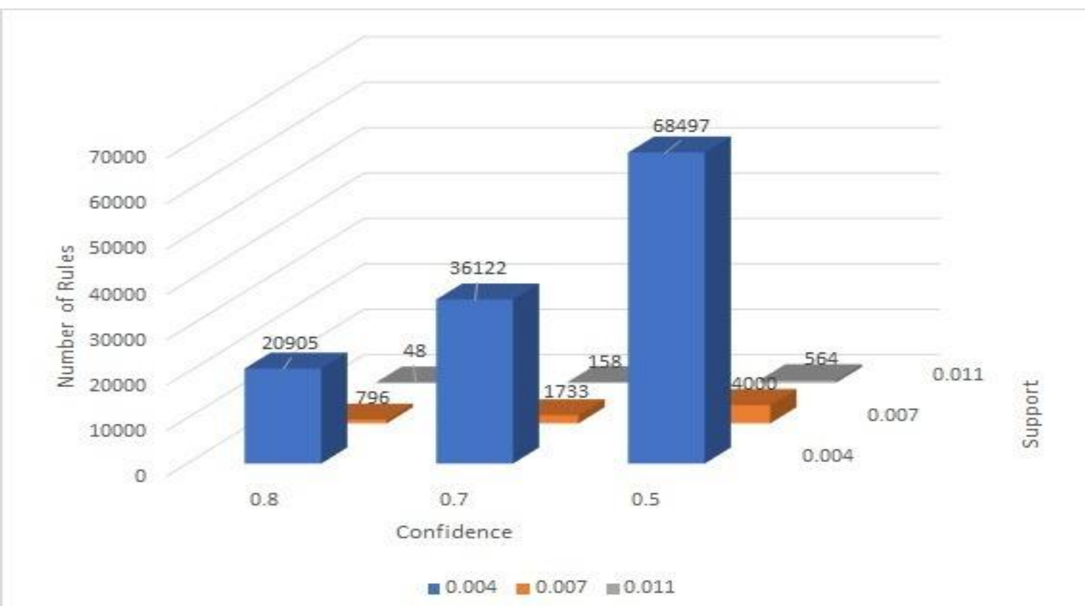


When the support is 0.007 (count = 140) we got 4014 items in candidate itemset and 1154 in frequent itemset after first iteration which changed to 143 for candidate itemset and 37 for frequent itemset by the end of the iteration 5.



When the support is 0.011 (count = 221) we got 4014 items in candidate itemset and 702 in frequent itemset after first iteration which changed to 1 for candidate itemset and 0 for frequent itemset by the end of the iteration 5.

<u>3. Generate rules for each specified min_conf (for each min_sup). Plot the number of rules generated with min_sup on X-axis and min_conf on Y-axis. The analysis report of the above is provided below.</u>



```
association_rules_1 <- apriori(tr, parameter = list(supp=0.004, conf=0.5,maxlen=10))
association_rules_2 <- apriori(tr, parameter = list(supp=0.004, conf=0.7,maxlen=10))
association_rules_3 <- apriori(tr, parameter = list(supp=0.004, conf=0.8,maxlen=10))
association_rules_4 <- apriori(tr, parameter = list(supp=0.007, conf=0.5,maxlen=10))
association_rules_5 <- apriori(tr, parameter = list(supp=0.007, conf=0.7,maxlen=10))
association_rules_6 <- apriori(tr, parameter = list(supp=0.007, conf=0.8,maxlen=10))
association_rules_7 <- apriori(tr, parameter = list(supp=0.011, conf=0.5,maxlen=10))
association_rules_8 <- apriori(tr, parameter = list(supp=0.011, conf=0.7,maxlen=10))
association_rules_9 <- apriori(tr, parameter = list(supp=0.011, conf=0.8,maxlen=10))
```

```
See help("Deprecated")
> inspect(association_rules_1)
     lhs       rhs    support     confidence coverage    lift       count
[1]  {598}  => {597}  0.004279459 0.7288136  0.005871815  88.765074  86
[2]  {597}  => {598}  0.004279459 0.5212121  0.008210589  88.765074  86
[3]  {2164} => {2517} 0.004179936 0.7241379  0.005772293 112.808340  84
[4]  {2517} => {2164} 0.004179936 0.6511628  0.006419188 112.808340  84
[5]  {2517} => {2016} 0.004677548 0.7286822  0.006419188 107.673507  94
[6]  {2016} => {2517} 0.004677548 0.6911765  0.006767516 107.673507  94
[7]  {1524} => {1517} 0.006518710 0.6036866  0.010798169  56.956275 131
[8]  {1517} => {1524} 0.006518710 0.6150235  0.010599124  56.956275 131
[9]  {1524} => {676}  0.006021099 0.5576037  0.010798169  33.350011 121
[10] {3520} => {3518} 0.004578025 0.5714286  0.008011545  20.803313  92
```

From the above graph we can analyze that confidence with 0.5 and support with 0.004 has the highest number of rules that is 68497 and the lowest is 48 which is for confidence 0.8 and support 0.011. Likewise, we have the count of all the number of rules for each combination of support and confidence.

Analyzing above rules we can say that:

598(lhs) => 597(rhs) [Support = 0.4%, Confidence = 72%]

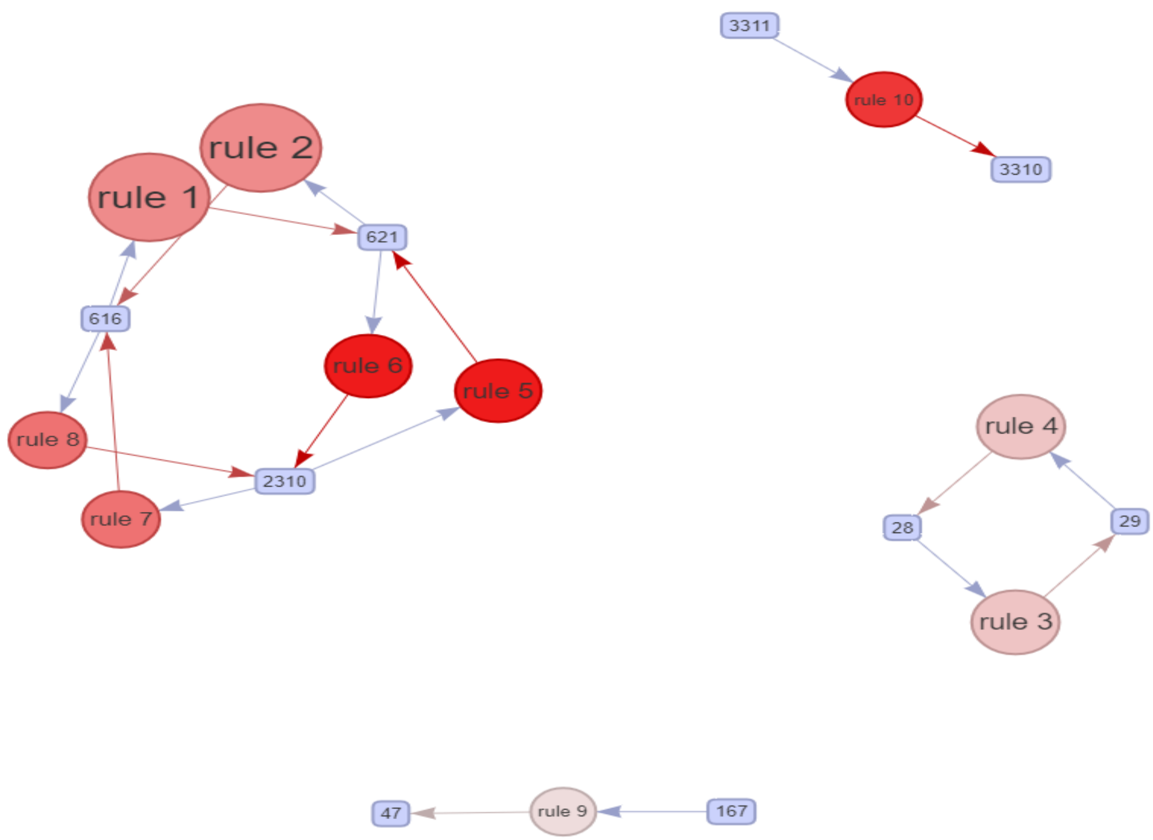
That is, 0.4% show transaction 597 is bought with purchase of 598 and 72% of customers who purchase 597 is bought with a purchase of 598.

4. Filter 10 rules each for lift > 10, Lift < 10

Greater than 10

```
association_rules_lift_more_10_1 <- subset(association_rules_1, subset = lift > 10)
top10<-head(sort(association_rules_lift_more_10_1),10)
plotly_arules(top10)
```

```
> inspect(top10)
     lhs       rhs    support    confidence coverage   lift     count
[1]  {616}  => {621}  0.03821656 0.7204503  0.05304538 14.26421 768
[2]  {621}  => {616}  0.03821656 0.7566502  0.05050756 14.26421 768
[3]  {28}   => {29}   0.03184713 0.6089439  0.05229896 12.48708 640
[4]  {29}   => {28}   0.03184713 0.6530612  0.04876592 12.48708 640
[5]  {2310} => {621}  0.03149881 0.8263708  0.03811704 16.36133 633
[6]  {621}  => {2310} 0.03149881 0.6236453  0.05050756 16.36133 633
[7]  {2310} => {616}  0.02980693 0.7819843  0.03811704 14.74180 599
[8]  {616}  => {2310} 0.02980693 0.5619137  0.05304538 14.74180 599
[9]  {167}  => {47}   0.02746815 0.6731707  0.04080414 11.66210 552
[10] {3311} => {3310} 0.02716959 0.7203166  0.03771895 15.85486 546
```

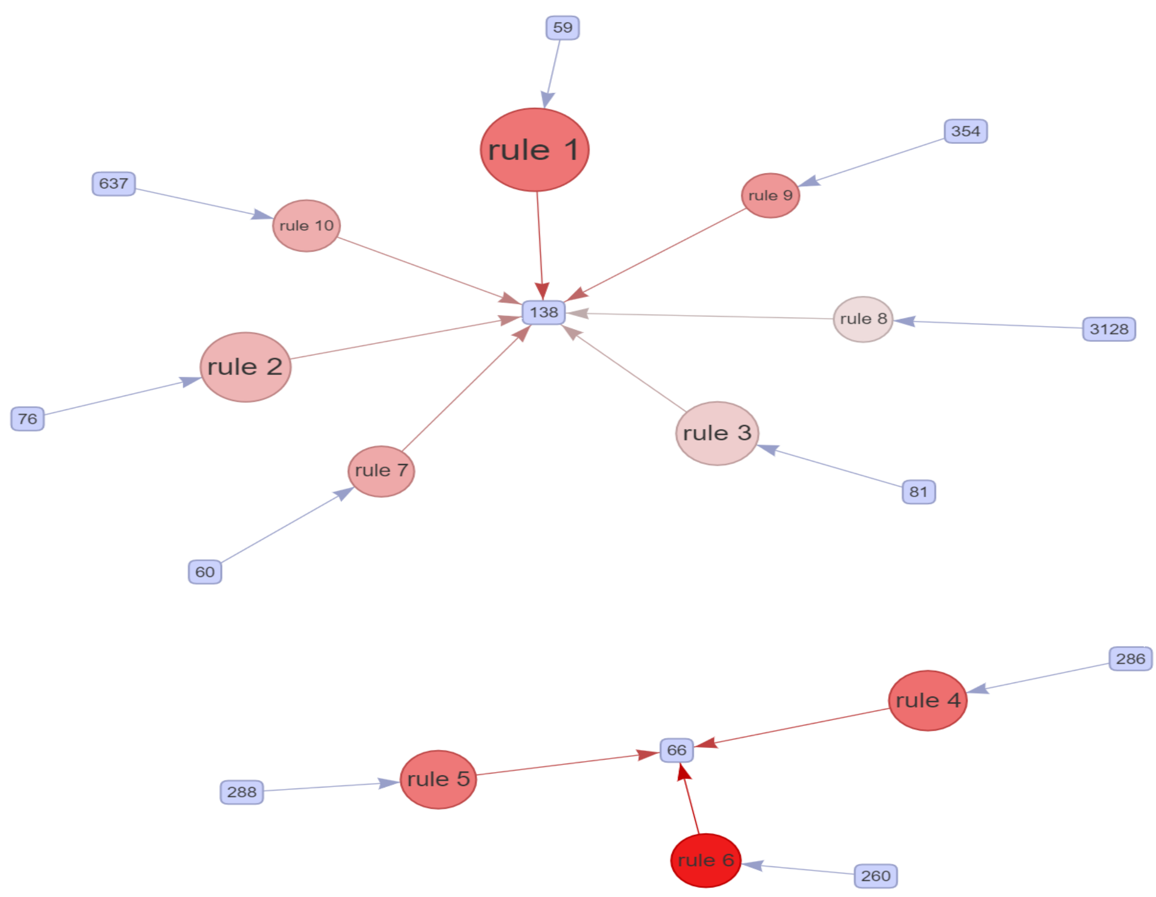


Graph based visualization of association rules uses vertices and edges where the vertices denote rules and edges are labelled which denote item name/description (in this graph integer value of item name/description). Blue arrows pointing from integer value indicates lhs and red arrow coming out from the rule indicates rhs. For example, item 616(lhs) points to rule 1 and that points to item 621(rhs). We can understand the other associations in the same manner.

Less than 10

```
top10_less<-head(sort(association_rules_lift_less_10_1),10)
plot(top10_less, method = "graph",  engine = "htmlwidget")
```

```
> inspect(top10_less)
     lhs       rhs   support    confidence coverage   lift     count
[1]  {59}   => {138} 0.04105295 0.6773399  0.06060908 6.506607 825
[2]  {76}   => {138} 0.03602707 0.6114865  0.05891720 5.874012 724
[3]  {81}   => {138} 0.03383758 0.5787234  0.05846935 5.559286 680
[4]  {286}  => {66}  0.03254379 0.5089494  0.06394307 6.539544 654
[5]  {288}  => {66}  0.03189689 0.5035350  0.06334594 6.469973 641
[6]  {260}  => {66}  0.03015525 0.5559633  0.05423965 7.143631 606
[7]  {60}   => {138} 0.02911027 0.6270096  0.04642715 6.023129 585
[8]  {3128} => {138} 0.02716959 0.5582822  0.04866640 5.362925 546
[9]  {354}  => {138} 0.02672174 0.6509091  0.04105295 6.252710 537
[10] {637}  => {138} 0.02662221 0.6199305  0.04294387 5.955126 535
```

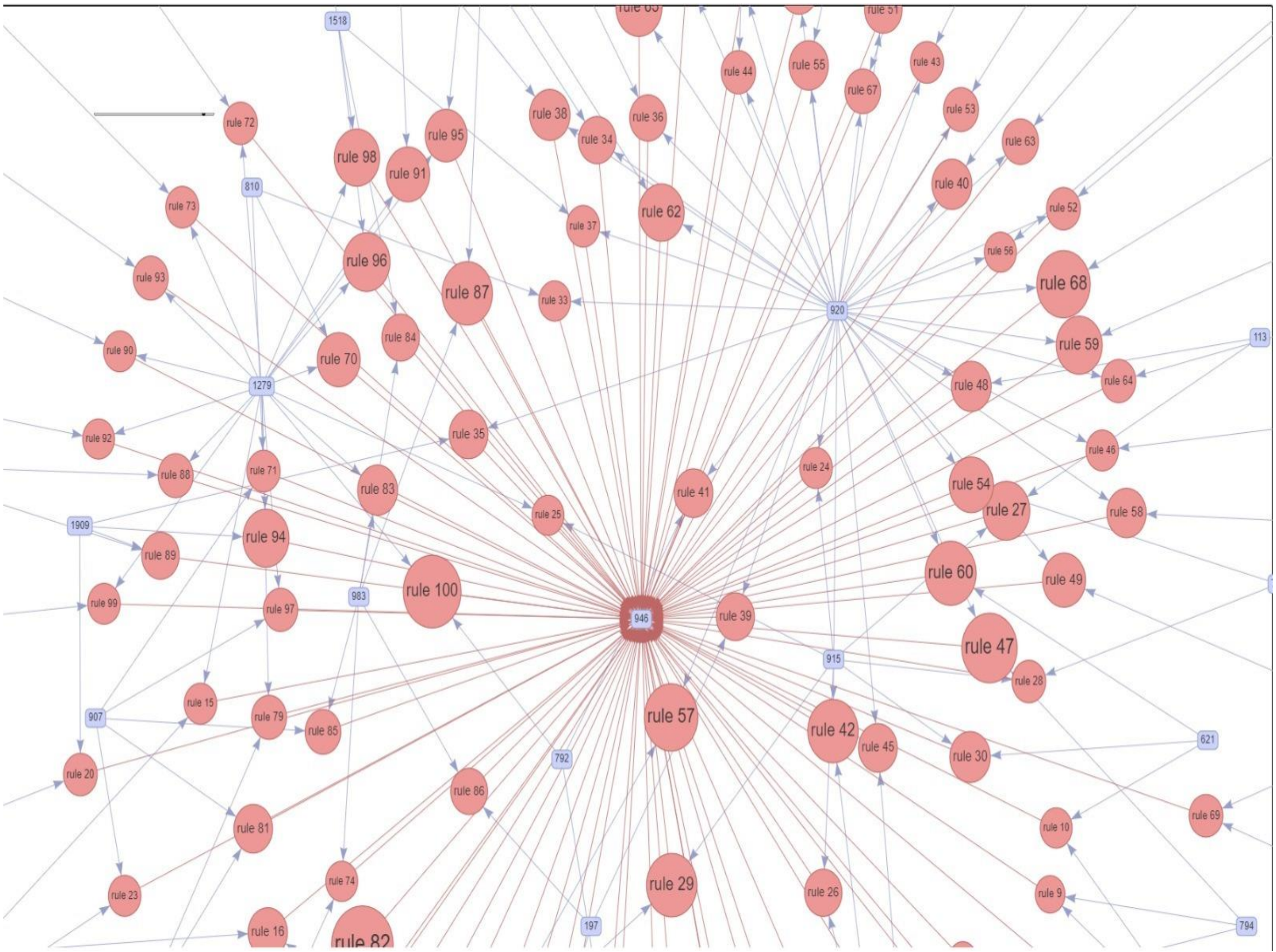


When we compare the above graphs for lift>10 and lift<10 we can analyze that items in lift>10 are more dependent to each other than items in lift<10.

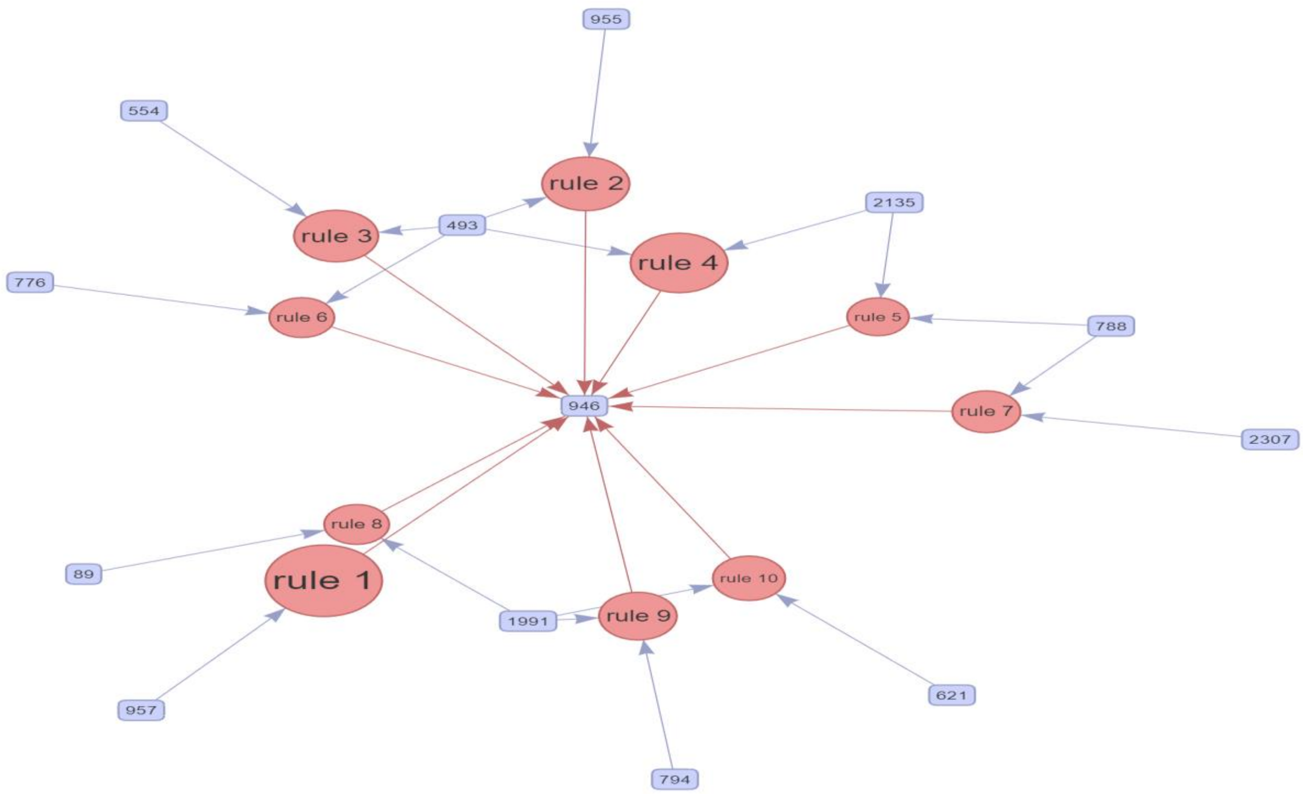
5. Visualize top 100 in descending order of min_con

We compared all the 9 combinations of support and confidence and we found that combinations from 1 to 3 are similar, 4 to 6 are similar, 7 and 8 are similar and 9 had 48 rules. In this report we are explaining combination 1 which has support 0.004 and confidence 0.5

```
> inspect(top100)
      lhs            rhs    support     confidence coverage    lift     count
[1]   {957}       => {946} 0.004876592 1          0.004876592 28.38418  98
[2]   {493,955}   => {946} 0.004428742 1          0.004428742 28.38418  89
[3]   {493,554}   => {946} 0.004378981 1          0.004378981 28.38418  88
[4]   {2135,493}  => {946} 0.004578025 1          0.004578025 28.38418  92
[5]   {2135,788}  => {946} 0.004030653 1          0.004030653 28.38418  81
[6]   {493,776}   => {946} 0.004080414 1          0.004080414 28.38418  82
[7]   {2307,788}  => {946} 0.004130175 1          0.004130175 28.38418  83
[8]   {1991,89}   => {946} 0.004080414 1          0.004080414 28.38418  82
[9]   {1991,794}  => {946} 0.004279459 1          0.004279459 28.38418  86
[10]  {1991,621}  => {946} 0.004030653 1          0.004030653 28.38418  81
[11]  {1310,429}  => {946} 0.004179936 1          0.004179936 28.38418  84
```



Plotting just the top 10 out of this will have a clearer picture as we can see that above graph is much congested.


955
554
rule 2
2135
rule 3
493
rule 4
776
rule 6
rule 5
788
946
rule 7
2307
rule 8
89
rule 1
rule 10
1991
rule 9
957
621
794