Clustering and Market Basket Analysis on Cincinnati Zoo data

Code

Hide

```
#Set working directory
getwd()
```

```
[1] "H:/"
```

Hide

```
setwd("H:/")
```

Association:

Support- It says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears.

Confidence- It says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears.

Lift- It says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.

Hide

```
#Load library for Association rule mining
library(arules)
#load data
assoc<-read.csv("food_4_association.csv")
#we dont need transaction id for modeling.
assoc<-assoc[,c(-1)]
#To load transactional data
assoc<-as(as.matrix(assoc), "transactions")
```

```
matrix contains values other than 0 and 1! Setting all entries != 0 to 1.
```

Hide

```
#Explore data
head(assoc[,1:6])
```

```
transactions in sparse format with
 6 transactions (rows) and
 6 items (columns)
```

Hide

```
dim(assoc)
```

```
[1] 19076   118
```

Hide

```
summary(assoc)
```

```
transactions as itemMatrix in sparse format with
 19076 rows (elements/itemsets/transactions) and
 118 columns (items) and a density of 0.02230729

most frequent items:
  Bottled.WaterFood Slice.of.CheeseFood    Medium.DrinkFood     Small.DrinkFood   Slice.of.PeppF
ood
               3166                3072                2871                2769                2
354
            (Other)
              35981

element (itemset/transaction) length distribution:
sizes
   0    1    2    3    4    5    6    7    8    9   10   11   12   13   15
 197 5675 5178 3253 2129 1293  655  351  178   95   42   14    8    7    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   1.000   2.000   2.632   4.000  15.000

includes extended item information - examples:
             labels
1    Add.CheeseFood
2          BeerFood
3 Bottled.WaterFood
```

Interpreting summary:

The density value of 0.02230729 (2.2 percent) refers to the proportion of nonzero matrix cells. Since there are 19076 * 118 = 2250968 positions in the matrix, we can calculate that a total of 2250968 * 0.02230729 = 50212.996 items were purchased.

Most frequent items shows: items that were most commonly found in the transactional data. Since 3166 / 118= 26.83, we can determine that bottled water appeared in 26.8 percent of the transactions.

A total of 5178 transactions contained only a single item, while 1 transaction had 15 items. The first quartile and median purchase sizes are 1 and 2 items, respectively, implying that 25 percent of the transactions contained 1 or less items. The mean of 2.632 items per transaction took place.

Hide

```
#Look at contents of sparse matrix
inspect(assoc[1:5])
```

```
    items
[1] {Bottled.WaterFood,
     Rice.Krispie.TreatFood,
     Sandwich.BasketFood,
     Slice.of.CheeseFood}
[2] {French.Fries.BasketFood,
     GatoradeFood,
     SandwichFood}
[3] {Soft.Pretzel..3_39Food,
     Souvenir.DrinkFood}
[4] {Small.DrinkFood}
[5] {Chicken.Tender.BasketFood}
```

Hide

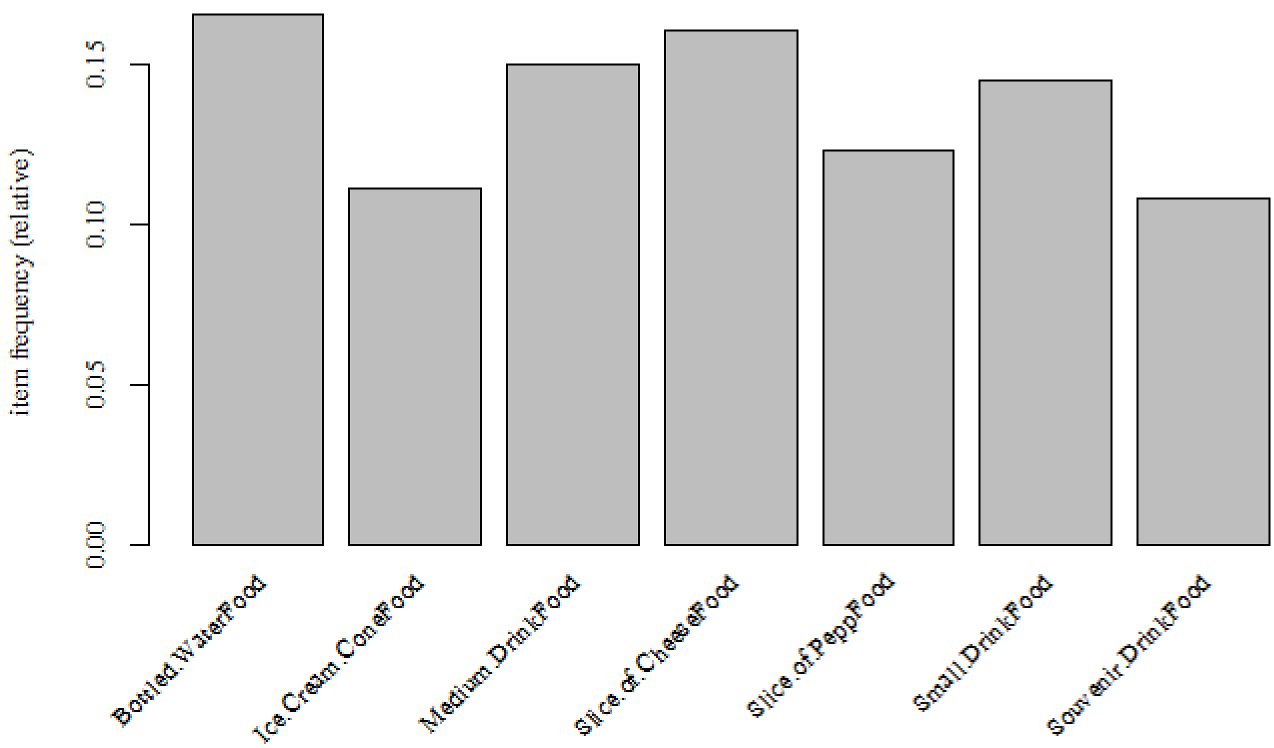
```
#To see the proportion of transactions that contain the item.
itemFrequency(assoc[,1:5])
```

```
           Add.CheeseFood                  BeerFood          Bottled.WaterFood
             0.0274690711              0.0135248480               0.1659677081
Bowl.of.Chili.w.CheeseFood         Bowl.of.ChiliFood
             0.0001572657              0.0001572657
```

It tells the proportion of transactions that contain the item.(Also known as Support)

Hide

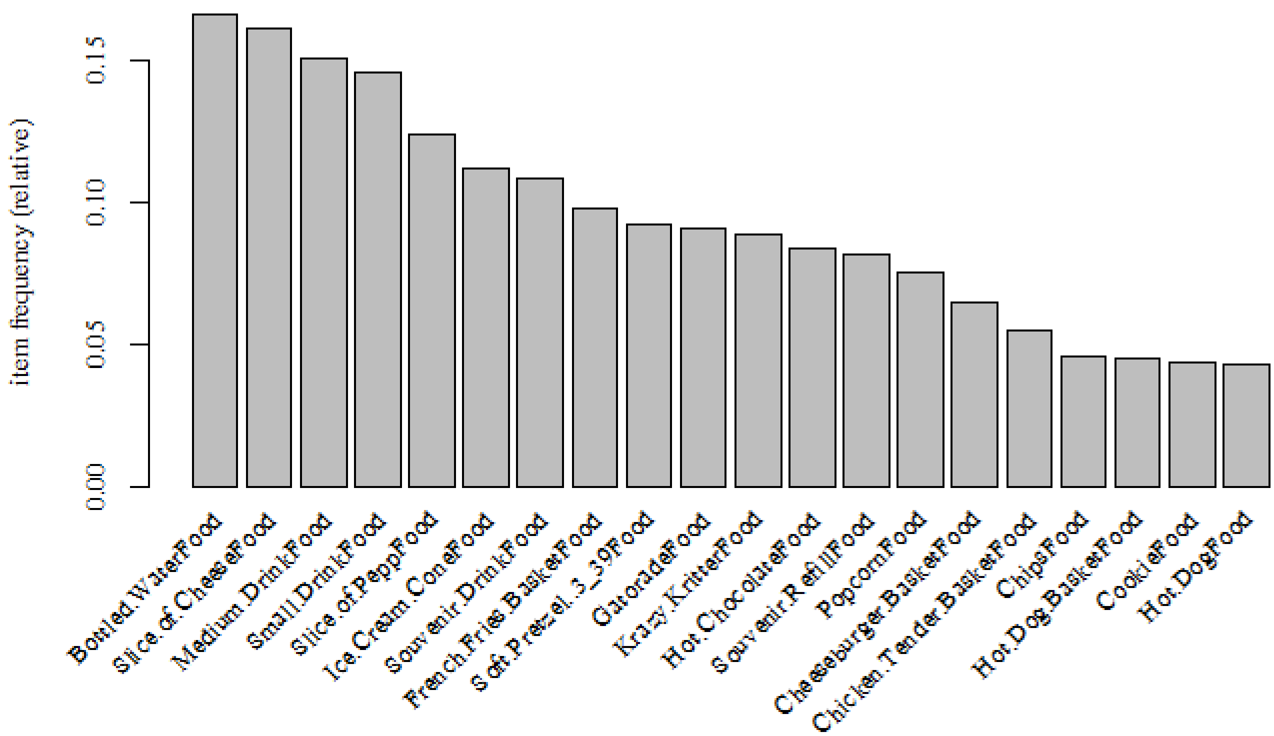
```
#Visualizing item support - item frequency plots
itemFrequencyPlot(assoc,support=0.1)
```



Here, Support means items that occurs 10% of transactions. We see that Bottled food,ice cream, drink food, cheese food etc. appeared in 10% of transactions. support = 0.1 means an item must have appeared in at least 0.1 * 19076 =1907 transactions.

Hide

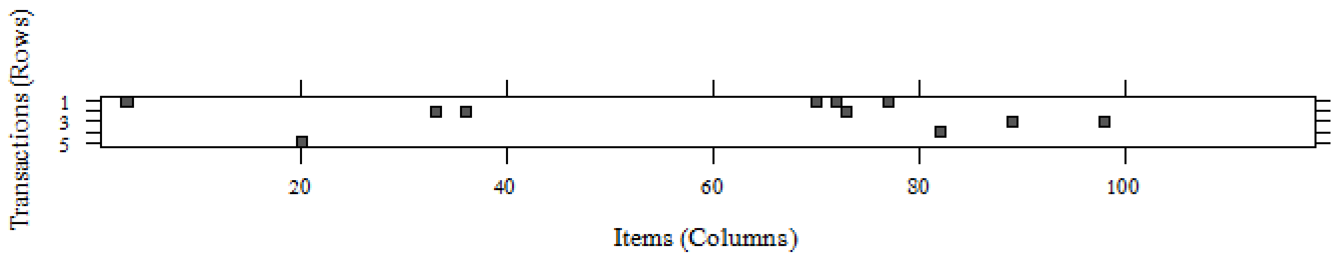
```
itemFrequencyPlot(assoc, topN = 20)
```



It shows top 20 items that are in transactions.

Hide

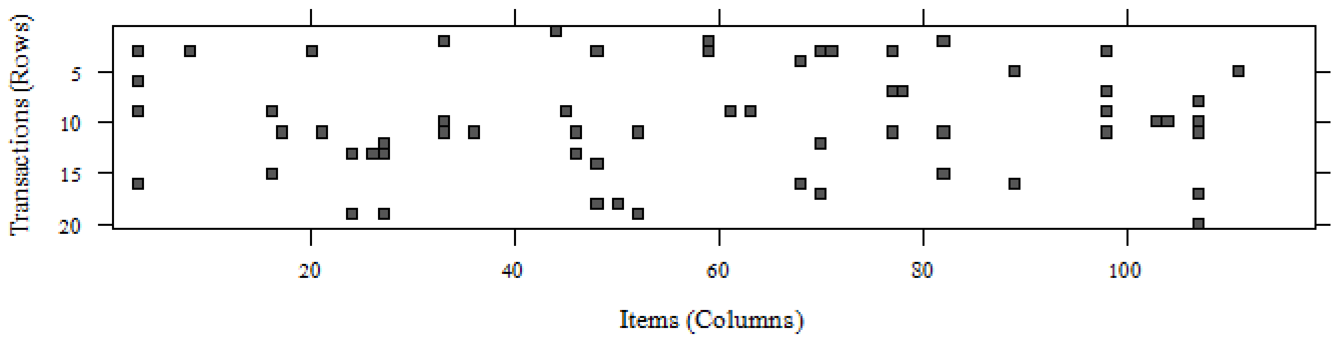
```
#Visualizing the transaction data - plotting the sparse matrix
image(assoc[1:5])
```



Cells in the matrix are filled with black for transactions (rows) where the item (column) was purchased. We can see that 1st transaction contains 4 items, 2nd-3 items and so on.

Hide

```
# for random 20 transactions
image(sample(assoc, 20))
```



Hide

```
#Fit model
rules <- apriori(assoc, parameter = list(support =0.005, confidence = 0.25, minlen = 2))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
       0.25    0.1    1 none FALSE            TRUE       5   0.005      2     10  rules FALSE

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

Absolute minimum support count: 95

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[115 item(s), 19076 transaction(s)] done [0.01s].
sorting and recoding items ... [58 item(s)] done [0.00s].
creating transaction tree ... done [0.02s].
checking subsets of size 1 2 3 4 done [0.01s].
writing ... [104 rule(s)] done [0.00s].
creating S4 object  ... done [0.01s].
```

minlen = 2 to eliminate rules that contain fewer than two items. confidence threshold of 0.25 means that in order to be included in the results, the rule has to be correct at least 25 percent of the time.

Hide

```
rules
```

```
set of 104 rules
```

We got 104 rules.

Hide

```
summary(rules)
```

```
set of 104 rules

rule length distribution (lhs + rhs):sizes
 2  3 
47 57 

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  2.000   2.000   3.000   2.548   3.000   3.000 

summary of quality measures:
    support           confidence          lift            count       
 Min.   :0.005032   Min.   :0.2500   Min.   : 1.601   Min.   :  96.0  
 1st Qu.:0.005924   1st Qu.:0.2737   1st Qu.: 2.093   1st Qu.: 113.0  
 Median :0.008283   Median :0.3214   Median : 2.919   Median : 158.0  
 Mean   :0.010979   Mean   :0.3955   Mean   : 3.937   Mean   : 209.4  
 3rd Qu.:0.011349   3rd Qu.:0.4275   3rd Qu.: 3.715   3rd Qu.: 216.5  
 Max.   :0.060862   Max.   :0.9982   Max.   :21.606   Max.   :1161.0  

mining info:
  data ntransactions support confidence
 assoc         19076   0.005       0.25
```

We see that 47 rules contains 2 items and 57 rules contain 3 items. lift of a rule measures how much more likely one item or itemset is purchased relative to its typical rate of purchase, given that we know another item or itemset has been purchased.

If lift is greater than one, it implies that the two items are found together more often than one would expect by chance. A large lift value is therefore a strong indicator that a rule is important, and reflects a true connectionbetween the items.

Hide

```
#Lets look at first 3 rules
inspect(rules[1:3])
```

```
    lhs                        rhs                          support     confidence lift      count
[1] {FloatFood}             => {Ice.Cream.ConeFood}         0.007024533 0.7089947   6.355631 134  
[2] {Side.of.CheeseFood} => {Hot.DogFood}                0.006290627 0.9230769  21.605663 120  
[3] {SandwichFood}          => {French.Fries.BasketFood} 0.007653596 0.6822430   6.989510 146  
```

1st rule tells us that this rule this rule covers 0.7 percent of the transactions and is correct in 70 percent of purchases involving float food. The lift value tells us how much more likely a customer is to buy whole Ice cream conefood relative to the average customer, given that he or she bought a float food.

Hide

```
#Improving model performance
#Sorting the set of association rules
inspect(sort(rules, by = "lift")[1:5])
```

```
    lhs                                            rhs                                support

[1] {Side.of.CheeseFood}                        => {Hot.DogFood}                      0.0062906
27
[2] {Hot.Chocolate.SouvenirFood}                => {Hot.Chocolate.Souvenir.RefillFood} 0.0149926
61
[3] {Hot.Chocolate.Souvenir.RefillFood}         => {Hot.Chocolate.SouvenirFood}       0.0149926
61
[4] {French.Fries.BasketFood,Krazy.KritterFood} => {Chicken.TendersFood}              0.0056615
64
[5] {Cheese.ConeyFood}                          => {Hot.DogFood}                      0.0111658
63
    confidence lift      count
[1] 0.9230769  21.605663 120
[2] 0.3530864  13.180972 286
[3] 0.5596869  13.180972 286
[4] 0.2523364  11.065678 108
[5] 0.4226190   9.891878 213
```

We see that people who buy hot dog food are nearly 21 times more likely to buy side of cheese food than the typical customer

Hide

```
#subset() function provides a method to search for subsets of transactions, items, or rules.
hot_dog_rules <- subset(rules, items %in% "Hot.DogFood")
inspect(hot_dog_rules)
```

```
    lhs                      rhs                          support     confidence lift      count
[1] {Side.of.CheeseFood} => {Hot.DogFood}                 0.006290627 0.9230769  21.605663 120
[2] {Cheese.ConeyFood}   => {Hot.DogFood}                 0.011165863 0.4226190   9.891878 213
[3] {Hot.DogFood}        => {Cheese.ConeyFood}            0.011165863 0.2613497   9.891878 213
[4] {Hot.DogFood}        => {French.Fries.BasketFood}     0.011585238 0.2711656   2.778064 221
[5] {Hot.DogFood}        => {Medium.DrinkFood}            0.010956175 0.2564417   1.703895 209
```

we can see that hot dog food is purchased frequently with side of cheese food, cheese coney food, french fries basketfood and medium drinkfood.

Hide

```
#rules containg hot dog food on left hand side
hot_dog_rules_lhs <- subset(rules, lhs %in% "Hot.DogFood")
hot_dog_rules_lhs
```

```
set of 3 rules
```

Hide

```
#rules containg hot dog food on right hand side
hot_dog_rules_rhs <- subset(rules, rhs %in% "Hot.DogFood")
hot_dog_rules_rhs
```

```
set of 2 rules
```

Hide

```
#rules matching either hot dog food or side of cheese food
hot_dog_cheese_food_rules<- subset(rules, items %in% c("Hot.DogFood","Side.of.CheeseFood"))
hot_dog_cheese_food_rules
```

```
set of 5 rules
```

Hide

```
#Partial matching allows us to find any item containing Chicken
chicken_rules<- subset(rules, items %pin% "Chicken")
chicken_rules
```

```
set of 13 rules
```

Hide

```
#rules having size >2
size<-subset(rules,size(rules)>2)
head(inspect(size))
```

```
     lhs                                               rhs                         support
[1]  {Chicken.TendersFood,Krazy.KritterFood}        => {French.Fries.BasketFood} 0.005661564
[2]  {Chicken.TendersFood,French.Fries.BasketFood} => {Krazy.KritterFood}       0.005661564
[3]  {French.Fries.BasketFood,Krazy.KritterFood}    => {Chicken.TendersFood}     0.005661564
[4]  {Chicken.TendersFood,French.Fries.BasketFood} => {Slice.of.CheeseFood}     0.005399455
[5]  {Chicken.TendersFood,Slice.of.CheeseFood}      => {French.Fries.BasketFood} 0.005399455
[6]  {CheeseburgerFood,Krazy.KritterFood}           => {French.Fries.BasketFood} 0.005451877
[7]  {CheeseburgerFood,French.Fries.BasketFood}     => {Krazy.KritterFood}       0.005451877
[8]  {CheeseburgerFood,French.Fries.BasketFood}     => {Medium.DrinkFood}        0.005189767
[9]  {CheeseburgerFood,Medium.DrinkFood}            => {French.Fries.BasketFood} 0.005189767
[10] {CheeseburgerFood,French.Fries.BasketFood}     => {Slice.of.CheeseFood}     0.005242189
[11] {CheeseburgerFood,Slice.of.CheeseFood}         => {French.Fries.BasketFood} 0.005242189
[12] {ChipsFood,Slice.of.PeppFood}                  => {Slice.of.CheeseFood}     0.008282659
[13] {ChipsFood,Slice.of.CheeseFood}                => {Slice.of.PeppFood}       0.008282659
[14] {Slice.of.PeppFood,Souvenir.RefillFood}        => {Slice.of.CheeseFood}     0.005347033
[15] {Slice.of.CheeseFood,Souvenir.RefillFood}      => {Slice.of.PeppFood}       0.005347033
[16] {GatoradeFood,Slice.of.PeppFood}               => {Slice.of.CheeseFood}     0.010117425
[17] {GatoradeFood,Slice.of.CheeseFood}             => {Slice.of.PeppFood}       0.010117425
[18] {French.Fries.BasketFood,Souvenir.DrinkFood}  => {Slice.of.CheeseFood}     0.005032502
[19] {Slice.of.CheeseFood,Souvenir.DrinkFood}       => {French.Fries.BasketFood} 0.005032502
[20] {Slice.of.PeppFood,Souvenir.DrinkFood}         => {Slice.of.CheeseFood}     0.008125393
[21] {Slice.of.CheeseFood,Souvenir.DrinkFood}       => {Slice.of.PeppFood}       0.008125393
[22] {French.Fries.BasketFood,Krazy.KritterFood}    => {Slice.of.PeppFood}       0.006028518
[23] {Krazy.KritterFood,Slice.of.PeppFood}          => {French.Fries.BasketFood} 0.006028518
[24] {French.Fries.BasketFood,Slice.of.PeppFood}    => {Krazy.KritterFood}       0.006028518
[25] {French.Fries.BasketFood,Krazy.KritterFood}    => {Small.DrinkFood}         0.005713986
[26] {Krazy.KritterFood,Small.DrinkFood}            => {French.Fries.BasketFood} 0.005713986
[27] {French.Fries.BasketFood,Small.DrinkFood}      => {Krazy.KritterFood}       0.005713986
[28] {French.Fries.BasketFood,Krazy.KritterFood}    => {Medium.DrinkFood}        0.006133361
[29] {Krazy.KritterFood,Medium.DrinkFood}           => {French.Fries.BasketFood} 0.006133361
[30] {French.Fries.BasketFood,Medium.DrinkFood}     => {Krazy.KritterFood}       0.006133361
[31] {French.Fries.BasketFood,Krazy.KritterFood}    => {Slice.of.CheeseFood}     0.009331097
[32] {Krazy.KritterFood,Slice.of.CheeseFood}        => {French.Fries.BasketFood} 0.009331097
[33] {French.Fries.BasketFood,Slice.of.CheeseFood} => {Krazy.KritterFood}       0.009331097
[34] {Krazy.KritterFood,Small.DrinkFood}            => {Slice.of.PeppFood}       0.005242189
[35] {Krazy.KritterFood,Slice.of.PeppFood}          => {Medium.DrinkFood}        0.005871252
[36] {Krazy.KritterFood,Medium.DrinkFood}           => {Slice.of.PeppFood}       0.005871252
[37] {Krazy.KritterFood,Slice.of.PeppFood}          => {Slice.of.CheeseFood}     0.011270707
[38] {Krazy.KritterFood,Slice.of.CheeseFood}        => {Slice.of.PeppFood}       0.011270707
[39] {Krazy.KritterFood,Small.DrinkFood}            => {Slice.of.CheeseFood}     0.007443909
[40] {Krazy.KritterFood,Medium.DrinkFood}           => {Slice.of.CheeseFood}     0.007024533
[41] {Bottled.WaterFood,Krazy.KritterFood}          => {Slice.of.CheeseFood}     0.006028518
[42] {French.Fries.BasketFood,Slice.of.PeppFood}    => {Small.DrinkFood}         0.006080939
[43] {French.Fries.BasketFood,Small.DrinkFood}      => {Slice.of.PeppFood}       0.006080939
[44] {French.Fries.BasketFood,Slice.of.PeppFood}    => {Slice.of.CheeseFood}     0.009068987
[45] {French.Fries.BasketFood,Slice.of.CheeseFood} => {Slice.of.PeppFood}       0.009068987
[46] {French.Fries.BasketFood,Small.DrinkFood}      => {Slice.of.CheeseFood}     0.008439925
[47] {French.Fries.BasketFood,Slice.of.CheeseFood} => {Small.DrinkFood}         0.008439925
[48] {French.Fries.BasketFood,Medium.DrinkFood}     => {Slice.of.CheeseFood}     0.006710002
[49] {Bottled.WaterFood,French.Fries.BasketFood}    => {Slice.of.CheeseFood}     0.005347033
[50] {Slice.of.PeppFood,Small.DrinkFood}            => {Slice.of.CheeseFood}     0.014258754
[51] {Slice.of.CheeseFood,Small.DrinkFood}          => {Slice.of.PeppFood}       0.014258754
[52] {Medium.DrinkFood,Slice.of.PeppFood}           => {Slice.of.CheeseFood}     0.013629692
```

```
[53] {Medium.DrinkFood,Slice.of.CheeseFood}        => {Slice.of.PeppFood}       0.013629692
[54] {Bottled.WaterFood,Slice.of.PeppFood}         => {Slice.of.CheeseFood}     0.010694066
[55] {Bottled.WaterFood,Slice.of.CheeseFood}       => {Slice.of.PeppFood}       0.010694066
[56] {Medium.DrinkFood,Small.DrinkFood}            => {Slice.of.CheeseFood}     0.006080939
[57] {Bottled.WaterFood,Small.DrinkFood}           => {Slice.of.CheeseFood}     0.005713986
     confidence lift      count
[1]  0.9557522   9.791584 108
[2]  0.3272727   3.694115 108
[3]  0.2523364  11.065678 108
[4]  0.3121212   1.938159 103
[5]  0.8728814   8.942580 103
[6]  0.8813559   9.029402 104
[7]  0.3229814   3.645676 104
[8]  0.3074534   2.042836  99
[9]  0.8761062   8.975619  99
[10] 0.3105590   1.928458 100
[11] 0.8695652   8.908607 100
[12] 0.5808824   3.607068 158
[13] 0.4730539   3.833465 158
[14] 0.4766355   2.959733 102
[15] 0.4214876   3.415589 102
[16] 0.5830816   3.620724 193
[17] 0.4837093   3.919812 193
[18] 0.3568773   2.216078  96
[19] 0.2539683   2.601879  96
[20] 0.5032468   3.124979 155
[21] 0.4100529   3.322927 155
[22] 0.2686916   2.177383 115
[23] 0.2649770   2.714662 115
[24] 0.3050398   3.443159 115
[25] 0.2546729   1.754475 109
[26] 0.2994505   3.067840 109
[27] 0.2589074   2.922436 109
[28] 0.2733645   1.816336 117
[29] 0.2846715   2.916431 117
[30] 0.2867647   3.236878 117
[31] 0.4158879   2.582512 178
[32] 0.3084922   3.160471 178
[33] 0.3345865   3.776669 178
[34] 0.2747253   2.226278 100
[35] 0.2580645   1.714677 112
[36] 0.2725061   2.208295 112
[37] 0.4953917   3.076202 215
[38] 0.3726170   3.019559 215
[39] 0.3901099   2.422440 142
[40] 0.3260341   2.024553 134
[41] 0.3453453   2.144469 115
[42] 0.3076923   2.119732 116
[43] 0.2755344   2.232836 116
[44] 0.4588859   2.849514 173
[45] 0.3251880   2.635211 173
[46] 0.3824228   2.374706 161
[47] 0.3026316   2.084868 161
[48] 0.3137255   1.948121 128
```

```
[49] 0.3238095   2.010739 102
[50] 0.4963504   3.082155 272
[51] 0.3788301   3.069908 272
[52] 0.5273834   3.274858 260
[53] 0.4421769   3.583248 260
[54] 0.5151515   3.198903 204
[55] 0.3669065   2.973283 204
[56] 0.2801932   1.739898 116
[57] 0.2738693   1.700629 109
                                              lhs                          rhs     support confi
dence
[1]       {Chicken.TendersFood,Krazy.KritterFood} => {French.Fries.BasketFood} 0.005661564  0.95
57522
[2] {Chicken.TendersFood,French.Fries.BasketFood} =>       {Krazy.KritterFood} 0.005661564  0.32
72727
[3]   {French.Fries.BasketFood,Krazy.KritterFood} =>     {Chicken.TendersFood} 0.005661564  0.25
23364
[4] {Chicken.TendersFood,French.Fries.BasketFood} =>     {Slice.of.CheeseFood} 0.005399455  0.31
21212
[5]     {Chicken.TendersFood,Slice.of.CheeseFood} => {French.Fries.BasketFood} 0.005399455  0.87
28814
[6]          {CheeseburgerFood,Krazy.KritterFood} => {French.Fries.BasketFood} 0.005451877  0.88
13559
         lift count
[1]  9.791584   108
[2]  3.694115   108
[3] 11.065678   108
[4]  1.938159   103
[5]  8.942580   103
[6]  9.029402   104
```

Hide

```
#rules having life >15
rule_lift<-subset(rules, lift>15)
inspect(rule_lift)
```

```
    lhs                      rhs           support     confidence lift     count
[1] {Side.of.CheeseFood} => {Hot.DogFood} 0.006290627 0.9230769  21.60566 120
```

Hide

```
#rules having confidence>0.50
rule_conf<-subset(rules, confidence>0.50)
rule_conf
```

```
set of 18 rules
```

Hide

```
head(inspect(rule_conf))
```

```
     lhs                                             rhs                             support     confi
dence
[1]  {FloatFood}                                  => {Ice.Cream.ConeFood}            0.007024533 0.708
9947
[2]  {Side.of.CheeseFood}                         => {Hot.DogFood}                   0.006290627 0.923
0769
[3]  {SandwichFood}                               => {French.Fries.BasketFood}       0.007653596 0.682
2430
[4]  {Hot.Chocolate.Souvenir.RefillFood}          => {Hot.Chocolate.SouvenirFood}    0.014992661 0.559
6869
[5]  {ToppingFood}                                => {Ice.Cream.ConeFood}            0.028569931 0.998
1685
[6]  {Add.CheeseFood}                             => {Soft.Pretzel..3_39Food}        0.019133990 0.696
5649
[7]  {Chicken.TendersFood}                        => {French.Fries.BasketFood}       0.017299224 0.758
6207
[8]  {CheeseburgerFood}                           => {French.Fries.BasketFood}       0.016879849 0.793
1034
[9]  {Chicken.TendersFood,Krazy.KritterFood}      => {French.Fries.BasketFood}       0.005661564 0.955
7522
[10] {Chicken.TendersFood,Slice.of.CheeseFood}    => {French.Fries.BasketFood}       0.005399455 0.872
8814
[11] {CheeseburgerFood,Krazy.KritterFood}         => {French.Fries.BasketFood}       0.005451877 0.881
3559
[12] {CheeseburgerFood,Medium.DrinkFood}          => {French.Fries.BasketFood}       0.005189767 0.876
1062
[13] {CheeseburgerFood,Slice.of.CheeseFood}       => {French.Fries.BasketFood}       0.005242189 0.869
5652
[14] {ChipsFood,Slice.of.PeppFood}                => {Slice.of.CheeseFood}           0.008282659 0.580
8824
[15] {GatoradeFood,Slice.of.PeppFood}             => {Slice.of.CheeseFood}           0.010117425 0.583
0816
[16] {Slice.of.PeppFood,Souvenir.DrinkFood}       => {Slice.of.CheeseFood}           0.008125393 0.503
2468
[17] {Medium.DrinkFood,Slice.of.PeppFood}         => {Slice.of.CheeseFood}           0.013629692 0.527
3834
[18] {Bottled.WaterFood,Slice.of.PeppFood}        => {Slice.of.CheeseFood}           0.010694066 0.515
1515
     lift      count
[1]   6.355631 134
[2]  21.605663 120
[3]   6.989510 146
[4]  13.180972 286
[5]   8.947868 545
[6]   7.601643 365
[7]   7.771992 330
[8]   8.125264 322
[9]   9.791584 108
[10]  8.942580 103
[11]  9.029402 104
[12]  8.975619  99
[13]  8.908607 100
[14]  3.607068 158
```

```
[15]  3.620724 193
[16]  3.124979 155
[17]  3.274858 260
[18]  3.198903 204
                                   lhs                                 rhs     support confidence
[1]                        {FloatFood} =>         {Ice.Cream.ConeFood} 0.007024533  0.7089947
[2]               {Side.of.CheeseFood} =>                {Hot.DogFood} 0.006290627  0.9230769
[3]                     {SandwichFood} =>    {French.Fries.BasketFood} 0.007653596  0.6822430
[4] {Hot.Chocolate.Souvenir.RefillFood} => {Hot.Chocolate.SouvenirFood} 0.014992661  0.5596869
[5]                      {ToppingFood} =>         {Ice.Cream.ConeFood} 0.028569931  0.9981685
[6]                   {Add.CheeseFood} =>     {Soft.Pretzel..3_39Food} 0.019133990  0.6965649
         lift count
[1]  6.355631   134
[2] 21.605663   120
[3]  6.989510   146
[4] 13.180972   286
[5]  8.947868   545
[6]  7.601643   365
```

Hide

```
#rules having lift>5 and item containing Hot Dog Food
rule_dog<-subset(rules,items %in% "Hot.DogFood" & lift>5)
rule_dog
```

```
set of 3 rules
```

Hide

```
inspect(rule_dog)
```

```
    lhs                     rhs                support     confidence lift      count
[1] {Side.of.CheeseFood} => {Hot.DogFood}      0.006290627 0.9230769  21.605663 120
[2] {Cheese.ConeyFood}   => {Hot.DogFood}      0.011165863 0.4226190   9.891878 213
[3] {Hot.DogFood}        => {Cheese.ConeyFood} 0.011165863 0.2613497   9.891878 213
```

Hide

```
#Saving association rules to a data frame
rules_dataframe <- as(rules, "data.frame")
head(rules_dataframe)
```

```
                                                               rules     support confidence
1                                  {FloatFood} => {Ice.Cream.ConeFood} 0.007024533  0.7089947
2                                {Side.of.CheeseFood} => {Hot.DogFood} 0.006290627  0.9230769
3                          {SandwichFood} => {French.Fries.BasketFood} 0.007653596  0.6822430
4 {Hot.Chocolate.Souvenir.RefillFood} => {Hot.Chocolate.SouvenirFood} 0.014992661  0.5596869
5 {Hot.Chocolate.SouvenirFood} => {Hot.Chocolate.Souvenir.RefillFood} 0.014992661  0.3530864
6                     {Burger.BasketFood} => {Cheeseburger.BasketFood} 0.005137345  0.2606383
       lift count
1  6.355631   134
2 21.605663   120
3  6.989510   146
4 13.180972   286
5 13.180972   286
6  4.042225    98
```

Hide

```
dim(rules_dataframe)
```

```
[1] 104   5
```

Hide

```
str(rules_dataframe)
```

```
'data.frame':   104 obs. of  5 variables:
 $ rules     : Factor w/ 104 levels "{Add.CheeseFood} => {Soft.Pretzel..3_39Food}",..: 36 93 92
60 61 7 8 63 9 104 ...
 $ support   : num  0.00702 0.00629 0.00765 0.01499 0.01499 ...
 $ confidence: num  0.709 0.923 0.682 0.56 0.353 ...
 $ lift      : num  6.36 21.61 6.99 13.18 13.18 ...
 $ count     : num  134 120 146 286 286 98 213 213 159 545 ...
```

Hide

```
summary(rules_dataframe)
```

```
                                                                 rules       support
 {Add.CheeseFood} => {Soft.Pretzel..3_39Food}                       : 1   Min.   :0.005032
 {Bottled.WaterFood,French.Fries.BasketFood} => {Slice.of.CheeseFood}: 1   1st Qu.:0.005924
 {Bottled.WaterFood,Krazy.KritterFood} => {Slice.of.CheeseFood}      : 1   Median :0.008283
 {Bottled.WaterFood,Slice.of.CheeseFood} => {Slice.of.PeppFood}      : 1   Mean   :0.010979
 {Bottled.WaterFood,Slice.of.PeppFood} => {Slice.of.CheeseFood}      : 1   3rd Qu.:0.011349
 {Bottled.WaterFood,Small.DrinkFood} => {Slice.of.CheeseFood}        : 1   Max.   :0.060862
 (Other)                                                            :98
   confidence          lift              count
 Min.   :0.2500   Min.   : 1.601   Min.   :  96.0
 1st Qu.:0.2737   1st Qu.: 2.093   1st Qu.: 113.0
 Median :0.3214   Median : 2.919   Median : 158.0
 Mean   :0.3955   Mean   : 3.937   Mean   : 209.4
 3rd Qu.:0.4275   3rd Qu.: 3.715   3rd Qu.: 216.5
 Max.   :0.9982   Max.   :21.606   Max.   :1161.0
```

Hide

```
#Saving association rules to a file
write.csv(rules_dataframe, file = "rules.csv")
```

Clustering: kmeans algorithm

Hide

```
#Load data
clustering1<-read.csv("qry_Food_by_Month.csv")
clustering<-clustering1
#we dont need nickname therefore removing it. k-means requires numeric variables.
clustering<-clustering[,2:7]
#Rename variables
names(clustering)<-c("Oct_10","Nov_10","Dec_10","Jan_11","Feb_11","Mar_11")
#structure of data
str(clustering)
```

```
'data.frame':   55 obs. of  6 variables:
 $ Oct_10: int  343 131 1448 188 32 37 662 529 395 74 ...
 $ Nov_10: int  66 79 410 86 2 55 274 106 299 6 ...
 $ Dec_10: int  99 232 577 103 0 59 292 15 298 3 ...
 $ Jan_11: int  37 12 59 19 0 3 51 0 61 7 ...
 $ Feb_11: int  4 18 165 40 1 33 93 0 132 16 ...
 $ Mar_11: int  105 49 507 73 0 65 266 72 298 14 ...
```

Hide

```
#load library
library(ggplot2)
#Explore data
head(clustering)
```

```
  Oct_10 Nov_10 Dec_10 Jan_11 Feb_11 Mar_11
1    343     66     99     37      4    105
2    131     79    232     12     18     49
3   1448    410    577     59    165    507
4    188     86    103     19     40     73
5     32      2      0      0      1      0
6     37     55     59      3     33     65
```

Hide

```
dim(clustering)
```

```
[1] 55  6
```

Hide

```
summary(clustering)
```

```
     Oct_10          Nov_10           Dec_10            Jan_11           Feb_11            Mar_11     
 Min.   :   0   Min.   :  2.0   Min.   :   0.0   Min.   :  0.0   Min.   :  0.00   Min.   :  0.0  
 1st Qu.:  39   1st Qu.: 26.0   1st Qu.:   5.5   1st Qu.:  0.0   1st Qu.:  0.00   1st Qu.: 15.0  
 Median : 154   Median : 66.0   Median :  56.0   Median :  8.0   Median : 11.00   Median : 48.0  
 Mean   : 371   Mean   :146.4   Mean   : 188.2   Mean   : 29.0   Mean   : 54.55   Mean   :150.6  
 3rd Qu.: 524   3rd Qu.:265.5   3rd Qu.: 275.0   3rd Qu.: 50.5   3rd Qu.: 93.50   3rd Qu.:232.5  
 Max.   :2002   Max.   :597.0   Max.   :1089.0   Max.   :186.0   Max.   :279.00   Max.   :785.0  
```

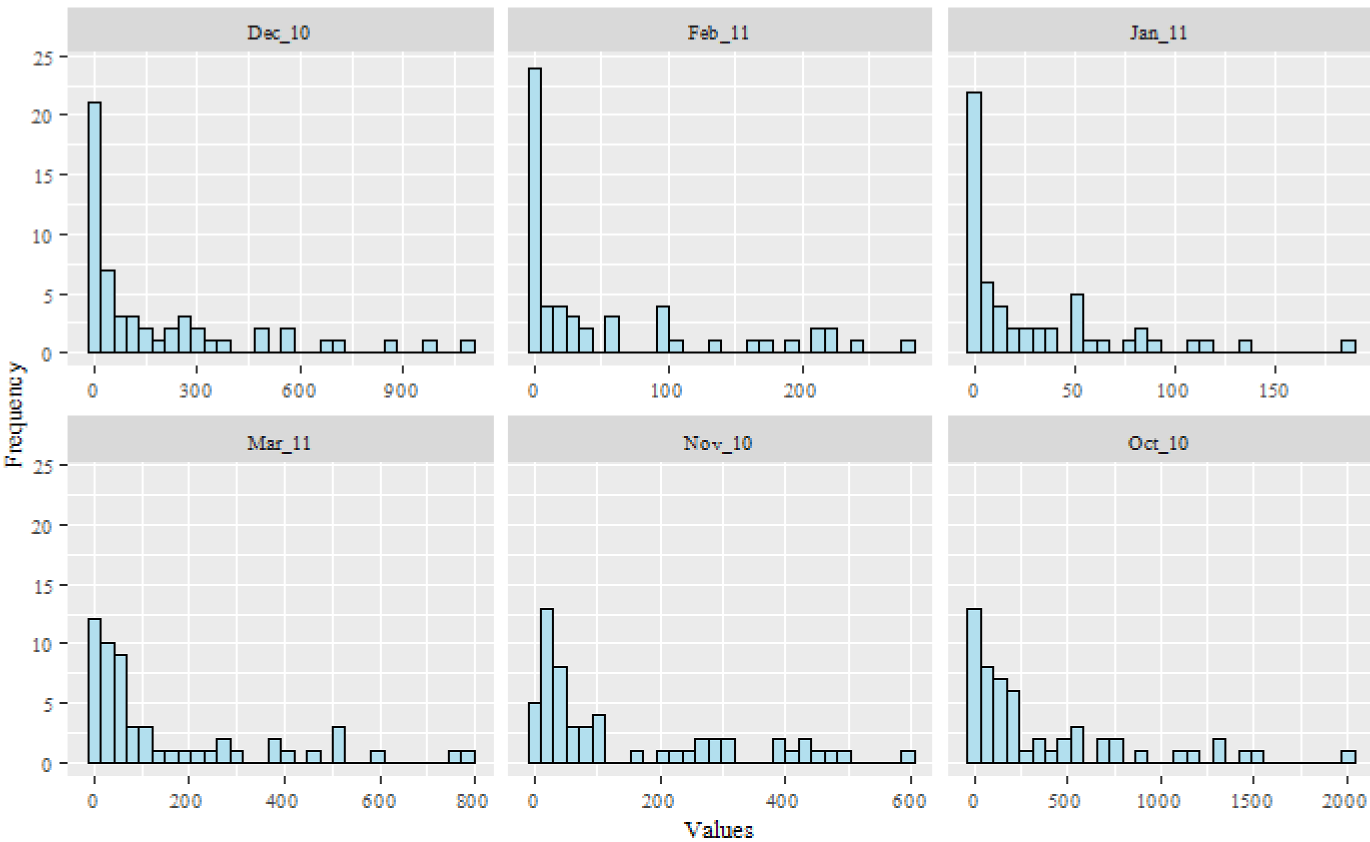
Hide

```
#Check for missing values
colSums(is.na(clustering))
```

```
Oct_10 Nov_10 Dec_10 Jan_11 Feb_11 Mar_11 
     0      0      0      0      0      0 
```

Hide

```
#Data analysis
library(tidyr)
# Histogram for each Attribute
clustering%>%
  gather(Attributes, value, 1:6) %>%
  ggplot(aes(x=value)) +
  geom_histogram(fill="lightblue2", colour="black") +
  facet_wrap(~Attributes, scales="free_x") +
  labs(x="Values", y="Frequency")
```



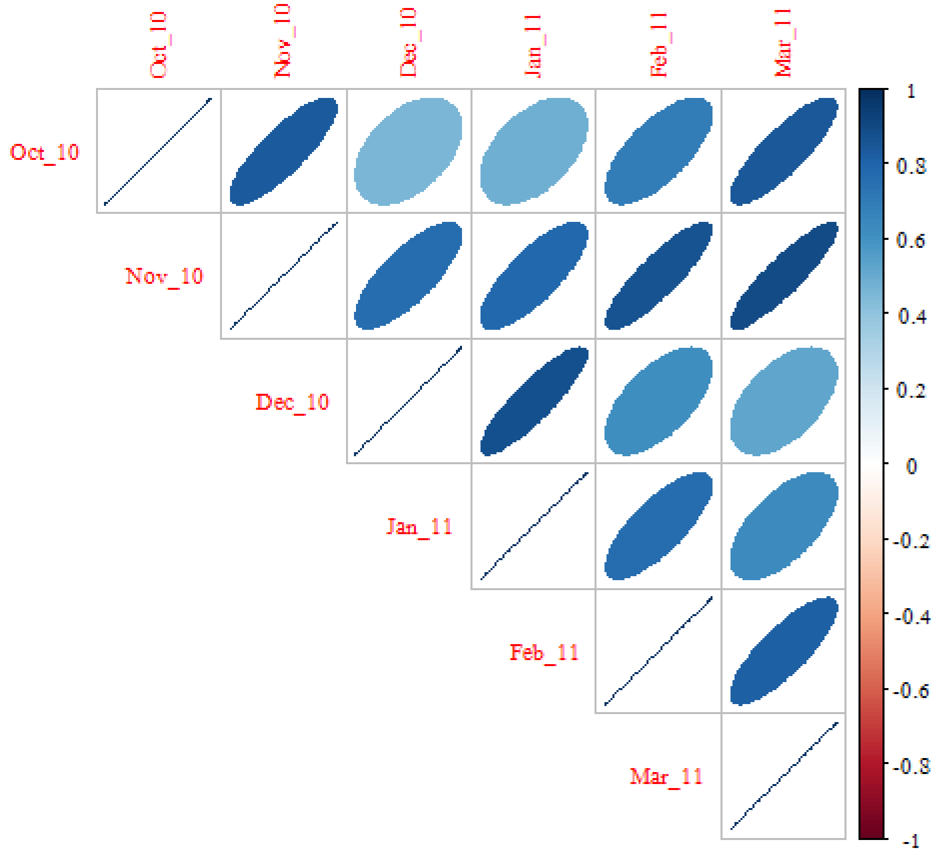
Hide

```
head(gather(clustering,Attributes,value,1:6))
```

```
  Attributes value
1     Oct_10   343
2     Oct_10   131
3     Oct_10  1448
4     Oct_10   188
5     Oct_10    32
6     Oct_10    37
```

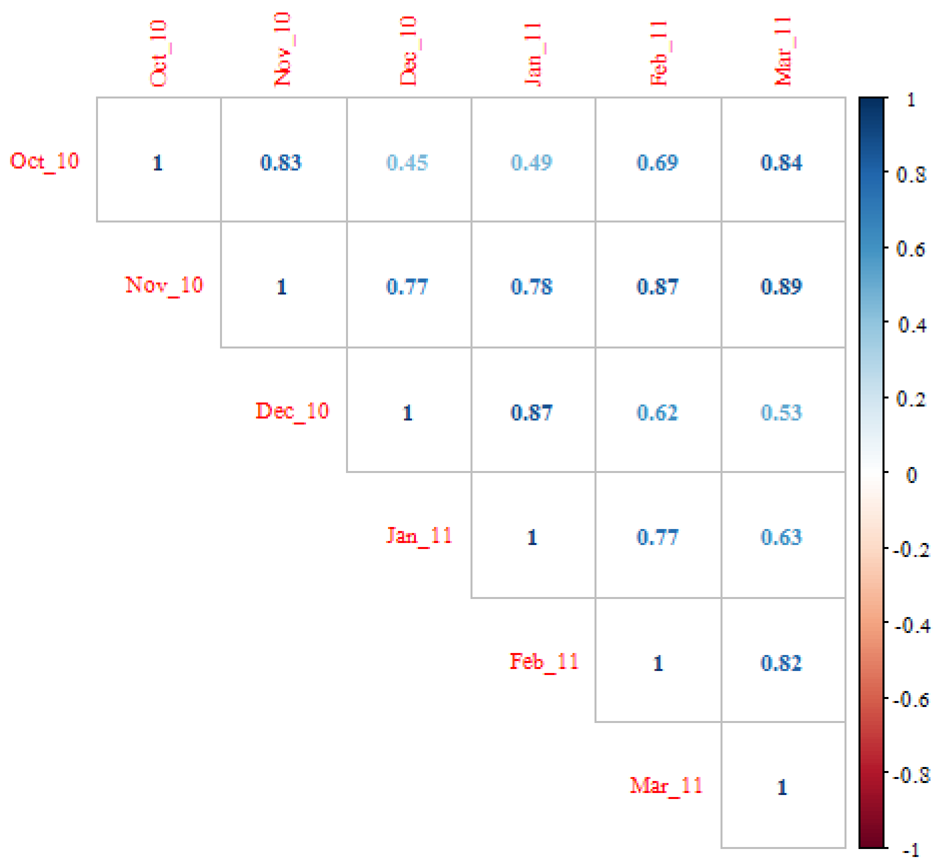
Hide

```
# Correlation matrix
library(corrplot)
corrplot(cor(clustering), type="upper", method="ellipse", tl.cex=0.9)
```



Hide

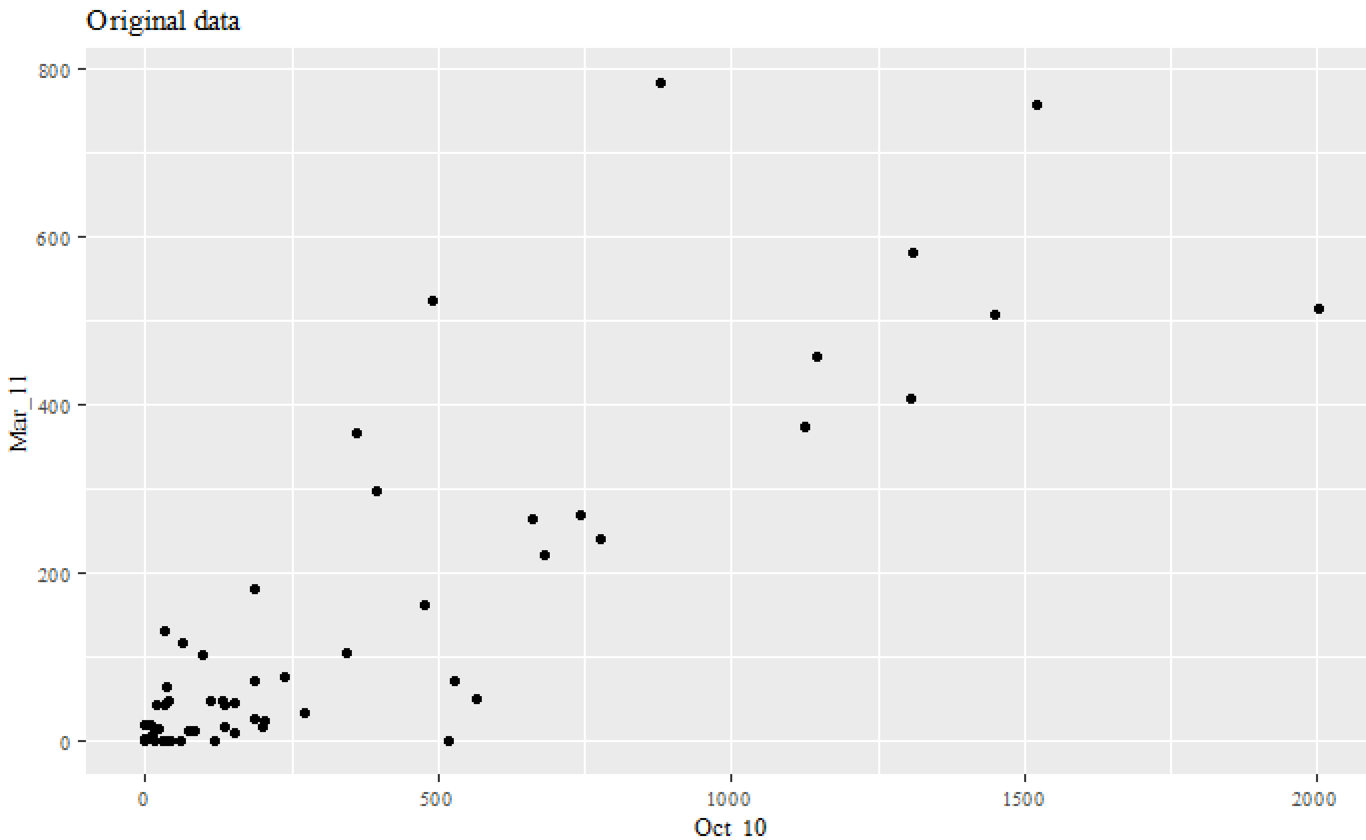
```
corrplot(cor(clustering), type="upper", method="number", tl.cex=0.9)
```



We see that transactions in Oct_10 has high correlation with Nov_10 and Mar_11. Feb_11 has high correlation with Mar_11 Dec_10 has high correlation with Jan_11 Nov_10 has high correlation with Feb_11 and Mar_11.

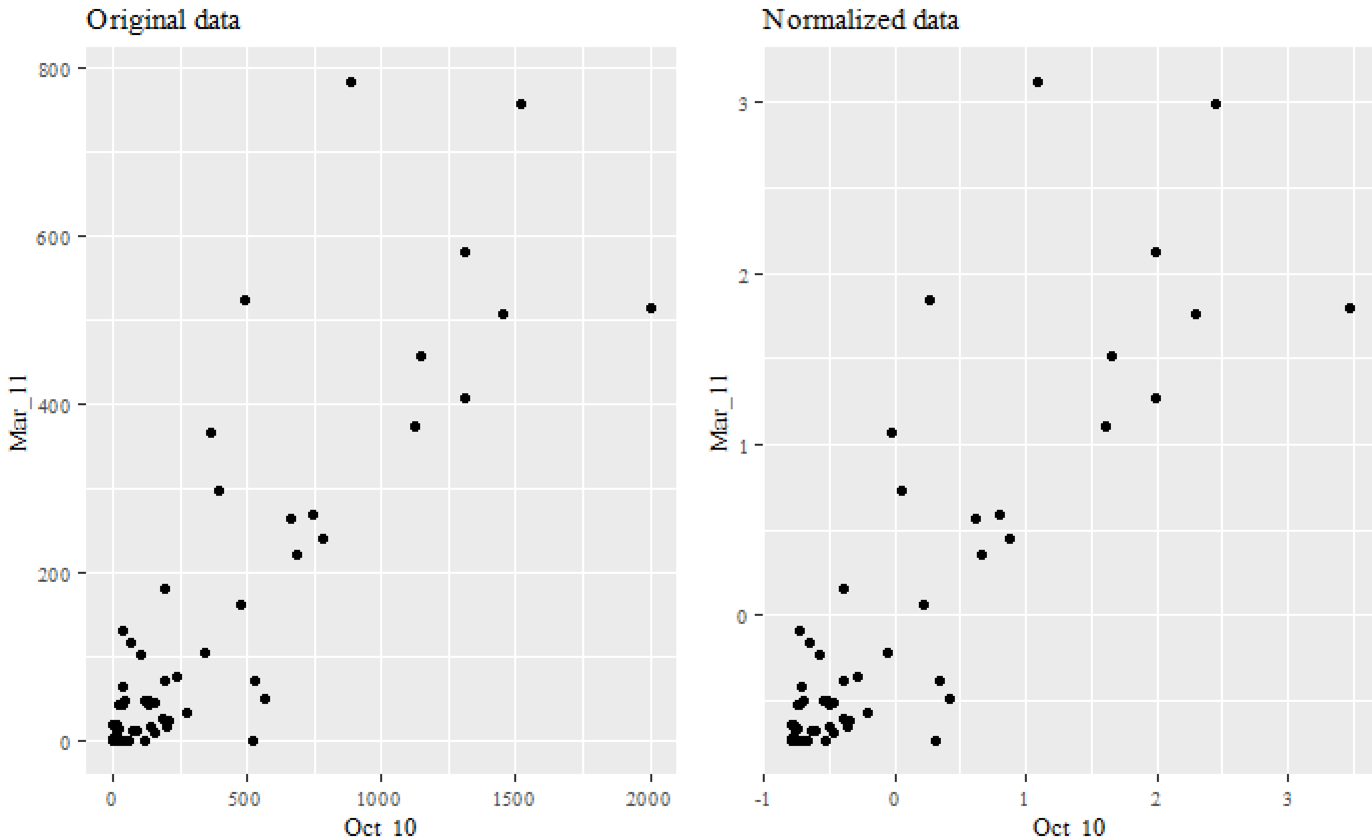
Hide

```
# Normalization
norm_data <- as.data.frame(scale(clustering))
# Original data
p1 <- ggplot(clustering, aes(x=Oct_10, y=Mar_11)) +
  geom_point() +
  labs(title="Original data")
p1
```



Hide

```
# Normalized data
p2 <- ggplot(norm_data, aes(x=Oct_10, y=Mar_11)) +
  geom_point() +
  labs(title="Normalized data")
# Subplot
library(gridExtra)
grid.arrange(p1, p2, ncol=2)
```



Hide

```
#k-means with k=3
set.seed(1234)
model <- kmeans(norm_data, centers=3)
model
```

```
K-means clustering with 3 clusters of sizes 10, 36, 9

Cluster means:
      Oct_10     Nov_10     Dec_10     Jan_11     Feb_11     Mar_11
1  1.6779610  1.7502256  1.0570593  1.2746319  1.7350387  1.8576906
2 -0.5016495 -0.6475860 -0.5418971 -0.5423530 -0.5500548 -0.5619770
3  0.1421967  0.6456491  0.9930780  0.7531544  0.2723983  0.1838072

Clustering vector:
 [1] 2 2 1 2 2 2 3 2 3 2 3 2 2 2 2 2 2 2 1 2 3 2 2 3 3 3 1 2 2 1 2 1 2 2 2 2 2 2 2 3 2 2 2 1 2 1
3 2 1
[50] 2 2 2 2 1 1

Within cluster sum of squares by cluster:
[1] 40.87337 12.57985 33.25827
 (between_SS / total_SS =  73.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

Hide

```
# Cluster to which each point is allocated
model$cluster
```

```
 [1] 2 2 1 2 2 2 3 2 3 2 3 2 2 2 2 2 2 2 1 2 3 2 2 3 3 3 1 2 2 1 2 1 2 2 2 2 2 2 2 3 2 2 2 1 2 1
3 2 1
[50] 2 2 2 2 1 1
```

Hide

```
# Cluster centers
model$centers
```

```
      Oct_10     Nov_10     Dec_10     Jan_11     Feb_11     Mar_11
1  1.6779610  1.7502256  1.0570593  1.2746319  1.7350387  1.8576906
2 -0.5016495 -0.6475860 -0.5418971 -0.5423530 -0.5500548 -0.5619770
3  0.1421967  0.6456491  0.9930780  0.7531544  0.2723983  0.1838072
```

Hide

```
# Cluster size
model$size
```

```
[1] 10 36  9
```

Hide

```
# Between-cluster sum of squares
model$betweenss
```

```
[1] 237.2885
```

Hide

```
# Within-cluster sum of squares
model$withinss
```

```
[1] 40.87337 12.57985 33.25827
```

Hide

```
# Total within-cluster sum of squares
model$tot.withinss
```

```
[1] 86.71149
```

Hide

```
# Total sum of squares
model$totss
```

```
[1] 324
```

Hide

```
#See which items are in which group
norm_data[model$cluster==1,]
```

```
        Oct_10   Nov_10     Dec_10     Jan_11     Feb_11   Mar_11
3   2.29395174 1.599730  1.4435463  0.7367815  1.4023896 1.755262
19  0.25768908 0.671271  0.3260383  1.1788503  2.3038433 1.838978
27  2.44731044 1.520842 -0.6986535 -0.7122221 -0.6925381 2.991304
30 -0.02559851 1.472295  0.3186130  1.3262066  2.0753057 1.065835
32  1.99362430 1.672551  1.0834192  1.3016473  1.4912653 1.267739
44  1.60809758 2.158019  1.0871318  1.5472411  1.7832855 1.100307
46  1.08625202 2.024516  1.8779265  1.9893100  1.9864300 3.124265
49  3.47396170 1.891012  0.6416137  0.6139846  2.1133953 1.799582
54  1.99575428 2.734514  2.4719508  2.6524133  2.8497941 2.124597
55  1.64856724 1.757508  2.0190073  2.1121069  2.0372161 1.509038
```

Hide

```
norm_data[model$cluster==2,]
```

```
       Oct_10     Nov_10     Dec_10      Jan_11      Feb_11      Mar_11
1  -0.05967822 -0.4877859 -0.3311010  0.19647506 -0.64175194 -0.22437693
2  -0.51123437 -0.4088972  0.1626816 -0.41750950 -0.46400050 -0.50014747
4  -0.38982541 -0.3664187 -0.3162504 -0.24559382 -0.18467682 -0.38196009
5  -0.72210258 -0.8761610 -0.6986535 -0.71222209 -0.67984153 -0.74144668
6  -0.71145267 -0.5545378 -0.4796070 -0.63854394 -0.27355254 -0.42135588
8   0.33649841 -0.2450514 -0.6429637 -0.71222209 -0.69253806 -0.38688457
10 -0.63264334 -0.8518875 -0.6875155 -0.54030641 -0.48939357 -0.67250405
12 -0.65607314 -0.2147096 -0.2754112  0.51574703  0.09464687 -0.16528325
13  0.31519859 -0.7062469 -0.6726649 -0.71222209 -0.69253806 -0.73652221
14 -0.61347350 -0.7851356 -0.6875155 -0.71222209 -0.69253806 -0.67742852
15 -0.35361572 -0.6394949 -0.6763776 -0.71222209 -0.69253806 -0.61833484
16 -0.49845448 -0.7608621 -0.6949408 -0.71222209 -0.69253806 -0.65280615
17 -0.66459307 -0.7851356 -0.6912282 -0.71222209 -0.69253806 -0.74144668
18 -0.54957405 -0.7183836 -0.6986535 -0.71222209 -0.69253806 -0.50507194
20 -0.79026200 -0.7001785  0.3111877 -0.44206888 -0.69253806 -0.74144668
22 -0.57939379 -0.2814616 -0.4461932 -0.29471259  0.03116421 -0.23422588
23 -0.28332632 -0.3178718 -0.4276299 -0.41750950 -0.31164213 -0.35733773
28 -0.72636254 -0.8579559 -0.6986535 -0.71222209 -0.69253806 -0.73652221
29  0.41530774 -0.6819734 -0.6986535 -0.71222209 -0.69253806 -0.49029852
31 -0.75831227 -0.8700926 -0.6244004 -0.68766270 -0.69253806 -0.74144668
33 -0.72210258 -0.4392390 -0.1157672 -0.07367815  0.09464687 -0.09141614
34 -0.39408537 -0.7547938 -0.6838029 -0.71222209 -0.69253806 -0.60848589
35 -0.50484443 -0.7062469 -0.5946992 -0.51574703 -0.66714500 -0.52969431
36 -0.21090693 -0.6212898 -0.6763776 -0.71222209 -0.69253806 -0.56909010
37 -0.36426563 -0.7001785 -0.6838029 -0.71222209 -0.69253806 -0.65280615
38 -0.74553238 -0.7426570 -0.5575727 -0.68766270 -0.65444847 -0.66265510
39 -0.70293274 -0.5848797 -0.4907450 -0.54030641 -0.36242825 -0.50507194
41 -0.76896218 -0.8154774 -0.6652396 -0.71222209 -0.67984153 -0.71189984
42 -0.70080276 -0.6880418 -0.6949408 -0.71222209 -0.69253806 -0.74144668
43 -0.46224479 -0.7123152 -0.6689523 -0.71222209 -0.69253806 -0.69220194
45 -0.46863474 -0.4695808 -0.5278715 -0.54030641 -0.56557275 -0.51492089
48 -0.79026200 -0.8700926 -0.6652396 -0.54030641 -0.61635888 -0.64295721
50 -0.77322214 -0.8033406 -0.6875155 -0.51574703 -0.55287622 -0.64295721
51 -0.75831227 -0.8215457 -0.6206878 -0.71222209 -0.69253806 -0.67250405
52 -0.72210258 -0.6516316 -0.5204462 -0.31927197 -0.41321438 -0.52969431
53 -0.53679416 -0.8215457 -0.6800902 -0.71222209 -0.69253806 -0.73652221
```

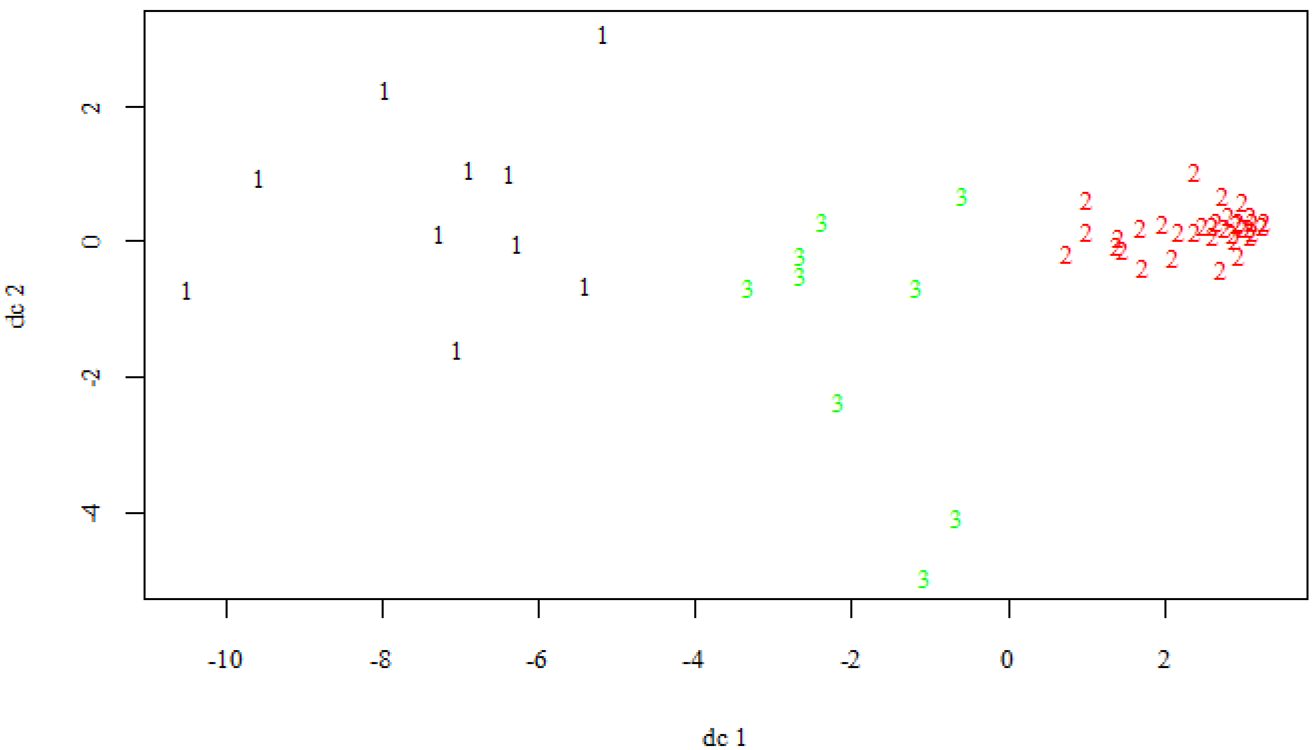
Hide

```
norm_data[model$cluster==3,]
```

```
       Oct_10    Nov_10      Dec_10      Jan_11     Feb_11      Mar_11
7   0.61978600 0.7744332  0.38544071  0.54030641  0.4882393  0.56846335
9   0.05108084 0.9261422  0.40771662  0.78590023  0.9834041  0.72604651
11  0.22147939 0.0887084 -0.14546843 -0.09823753  0.5263289  0.05631807
21  0.79444451 0.5499038  0.53394676  0.58942517  0.5009359  0.59308572
24 -0.75192233 0.4528101  3.34442381  3.85582301 -0.2227664 -0.51984536
25 -0.79026200 0.9322106  2.89519300  0.07367815 -0.4640005 -0.72667326
26  0.86686389 0.8047750  0.15525633  0.14735629  0.6532942  0.45027598
40 -0.38982541 0.3981948 -0.03037624  0.29471259  0.5390255  0.14988307
47  0.65812567 0.8836637  1.39156922  0.58942517 -0.5528762  0.35671097
```

Hide

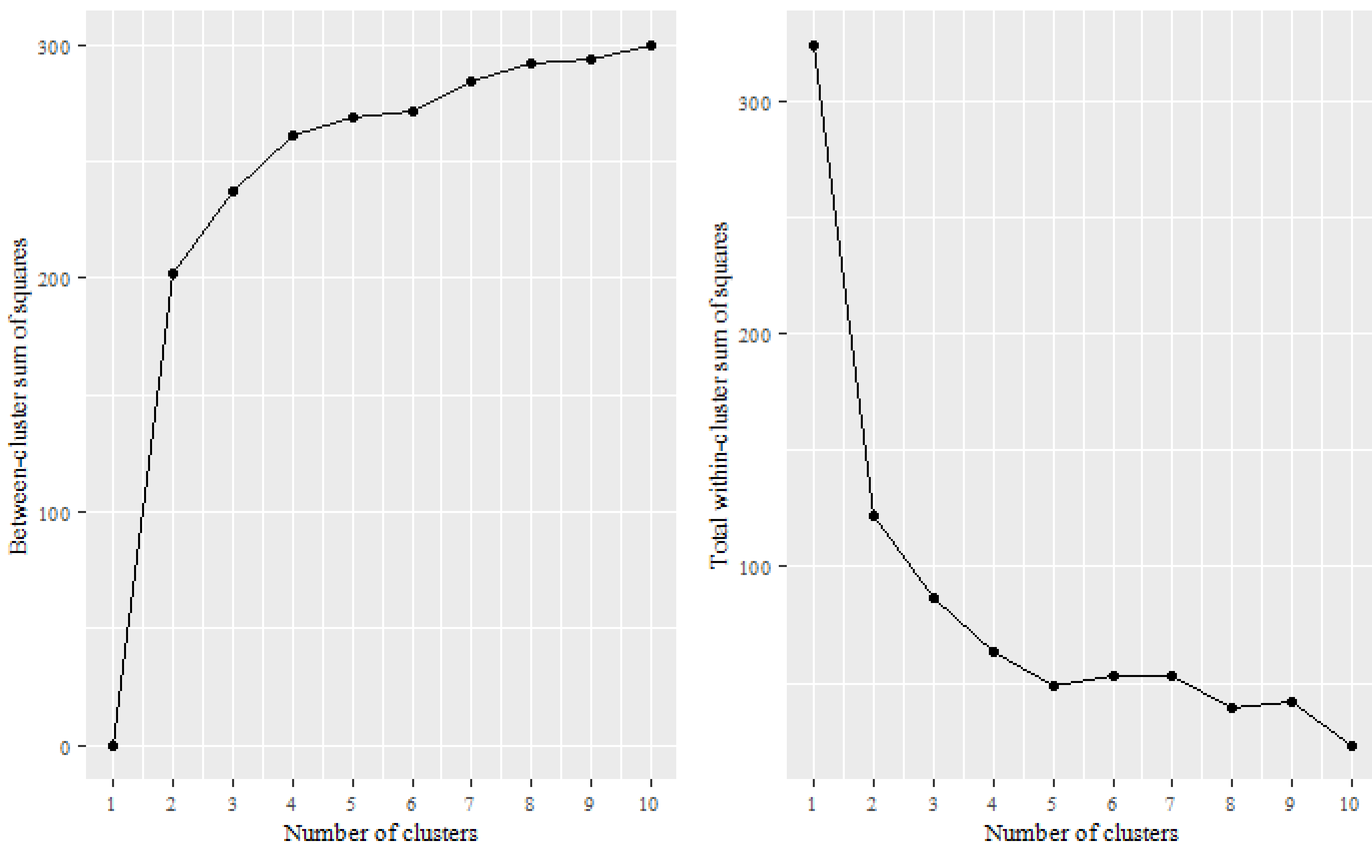
```
#plot clusters in k-means
library(fpc)
plotcluster(norm_data, model$cluster)
```



Hide

```
#To find elbow point to find better the value of k
bsss <- numeric()
wsss <- numeric()
# Run the algorithm for different values of k
set.seed(12345)
for(i in 1:10){

  # For each k, calculate betweenss and tot.withinss
  bsss[i] <- kmeans(norm_data, centers=i)$betweenss
  wsss[i] <- kmeans(norm_data, centers=i)$tot.withinss
}
# Between-cluster sum of squares vs Choice of k
p3 <- qplot(1:10, bsss, geom=c("point", "line"),
            xlab="Number of clusters", ylab="Between-cluster sum of squares") +
  scale_x_continuous(breaks=seq(0, 10, 1))
# Total within-cluster sum of squares vs Choice of k
p4 <- qplot(1:10, wsss, geom=c("point", "line"),
            xlab="Number of clusters", ylab="Total within-cluster sum of squares") +
  scale_x_continuous(breaks=seq(0, 10, 1))
#Using elbow method, we see that there should be 2 clusters.
# Subplot
grid.arrange(p3, p4, ncol=2)
```



We see that elbow occurs at k=2. So, we'll now apply kmeans with k=2.

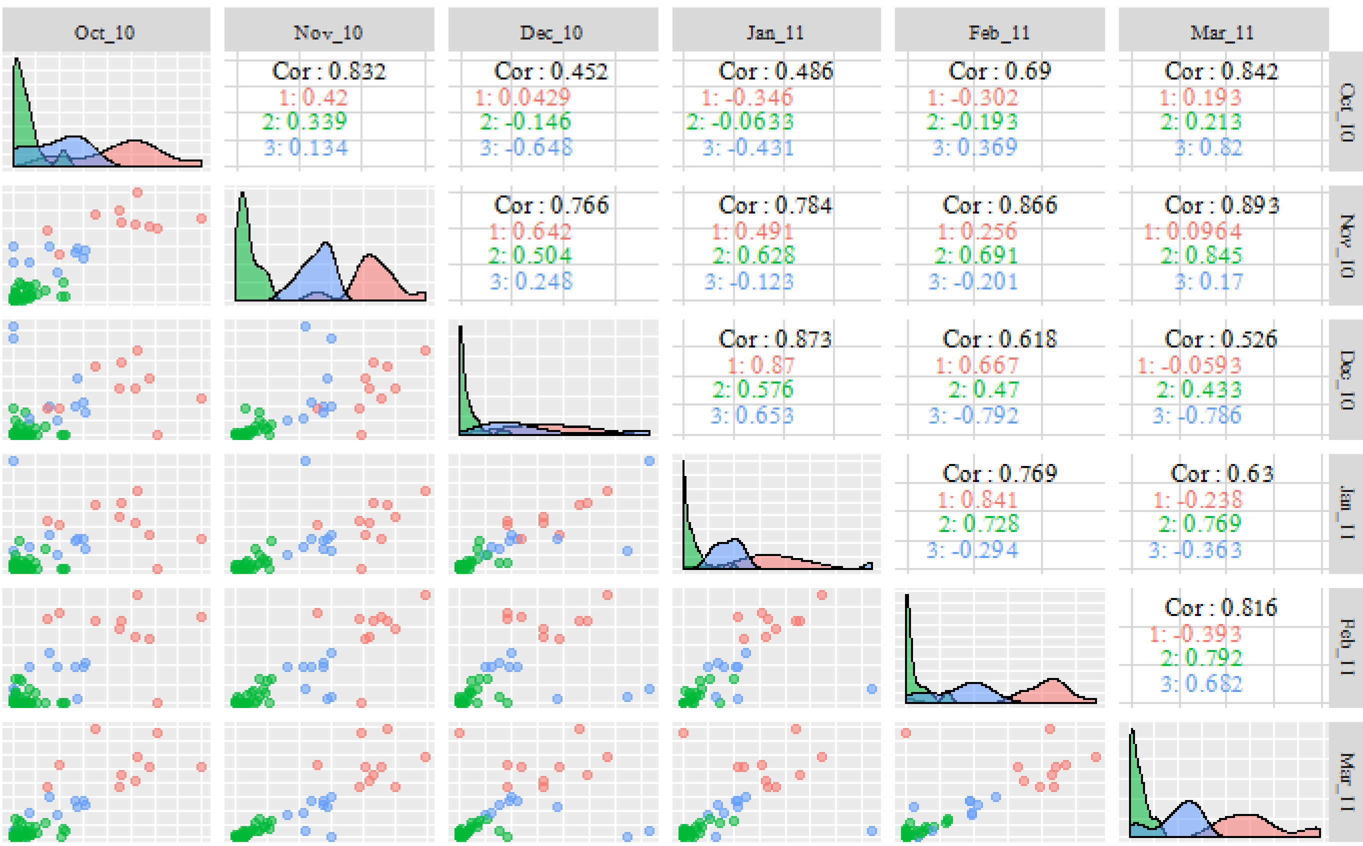
Hide

```
# Mean values of each cluster
aggregate(clustering, by=list(model$cluster), mean)
```

```
  Group.1    Oct_10    Nov_10    Dec_10    Jan_11    Feb_11    Mar_11
1       1 1158.8000 434.80000 472.90000 80.900000 191.20000 527.80000
2       2  135.5000  39.66667  42.22222  6.916667  11.22222  36.44444
3       3  437.7778 252.77778 455.66667 59.666667  76.00000 187.88889
```

Hide

```
library(GGally)
#different plots to be combined into a plot matrix
ggpairs(cbind(clustering, Cluster=as.factor(model$cluster)),
        columns=1:6, aes(colour=Cluster, alpha=0.5),
        lower=list(continuous="points"),
        axisLabels="none")
```



Hide

```
#k-means with k=2
set.seed(12345)
model1 <- kmeans(norm_data, centers=2)
model1
```

```
K-means clustering with 2 clusters of sizes 16, 39

Cluster means:
      Oct_10     Nov_10     Dec_10     Jan_11     Feb_11     Mar_11
1  1.1886243  1.3683740  1.0493092  1.2034097  1.2000386  1.2969777
2 -0.4876407 -0.5613842 -0.4304858 -0.4937066 -0.4923235 -0.5320934

Clustering vector:
 [1] 2 2 1 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 2 1 2 2 1 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1
1 2 1
[50] 2 2 2 2 1 1

Within cluster sum of squares by cluster:
[1] 88.82912 33.06954
 (between_SS / total_SS =  62.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

Hide

```
# Cluster to which each point is allocated
model1$cluster
```

```
 [1] 2 2 1 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 2 1 2 2 1 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1
1 2 1
[50] 2 2 2 2 1 1
```

Hide

```
# Cluster centers
model1$centers
```

```
      Oct_10     Nov_10     Dec_10     Jan_11     Feb_11     Mar_11
1  1.1886243  1.3683740  1.0493092  1.2034097  1.2000386  1.2969777
2 -0.4876407 -0.5613842 -0.4304858 -0.4937066 -0.4923235 -0.5320934
```

Hide

```
# Cluster size
model1$size
```

```
[1] 16 39
```

Hide

```
# Between-cluster sum of squares
model1$betweenss
```

```
[1] 202.1013
```

Hide

```
# Within-cluster sum of squares
model1$withinss
```

```
[1] 88.82912 33.06954
```

Hide

```
# Total within-cluster sum of squares
model1$tot.withinss
```

```
[1] 121.8987
```

Hide

```
# Total sum of squares
model1$totss
```

```
[1] 324
```

Hide

```
#See which items are in which group
norm_data[model1$cluster==1,]
```

```
         Oct_10    Nov_10     Dec_10     Jan_11     Feb_11     Mar_11
3    2.29395174 1.5997303  1.4435463  0.7367815  1.4023896  1.7552615
7    0.61978600 0.7744332  0.3854407  0.5403064  0.4882393  0.5684633
9    0.05108084 0.9261422  0.4077166  0.7859002  0.9834041  0.7260465
19   0.25768908 0.6712710  0.3260383  1.1788503  2.3038433  1.8389776
21   0.79444451 0.5499038  0.5339468  0.5894252  0.5009359  0.5930857
24  -0.75192233 0.4528101  3.3444238  3.8558230 -0.2227664 -0.5198454
26   0.86686389 0.8047750  0.1552563  0.1473563  0.6532942  0.4502760
27   2.44731044 1.5208416 -0.6986535 -0.7122221 -0.6925381  2.9913045
30  -0.02559851 1.4722947  0.3186130  1.3262066  2.0753057  1.0658352
32   1.99362430 1.6725506  1.0834192  1.3016473  1.4912653  1.2677386
44   1.60809758 2.1580194  1.0871318  1.5472411  1.7832855  1.1003065
46   1.08625202 2.0245155  1.8779265  1.9893100  1.9864300  3.1242652
47   0.65812567 0.8836637  1.3915692  0.5894252 -0.5528762  0.3567110
49   3.47396170 1.8910116  0.6416137  0.6139846  2.1133953  1.7995818
54   1.99575428 2.7345137  2.4719508  2.6524133  2.8497941  2.1245971
55   1.64856724 1.7575076  2.0190073  2.1121069  2.0372161  1.5090378
```

Hide

```
norm_data[model1$cluster==2,]
```

```
        Oct_10     Nov_10      Dec_10      Jan_11      Feb_11      Mar_11
1  -0.05967822 -0.4877859 -0.33110100  0.19647506 -0.64175194 -0.22437693
2  -0.51123437 -0.4088972  0.16268163 -0.41750950 -0.46400050 -0.50014747
4  -0.38982541 -0.3664187 -0.31625039 -0.24559382 -0.18467682 -0.38196009
5  -0.72210258 -0.8761610 -0.69865348 -0.71222209 -0.67984153 -0.74144668
6  -0.71145267 -0.5545378 -0.47960705 -0.63854394 -0.27355254 -0.42135588
8   0.33649841 -0.2450514 -0.64296371 -0.71222209 -0.69253806 -0.38688457
10 -0.63264334 -0.8518875 -0.68751552 -0.54030641 -0.48939357 -0.67250405
11  0.22147939  0.0887084 -0.14546843 -0.09823753  0.52632893  0.05631807
12 -0.65607314 -0.2147096 -0.27541123  0.51574703  0.09464687 -0.16528325
13  0.31519859 -0.7062469 -0.67266492 -0.71222209 -0.69253806 -0.73652221
14 -0.61347350 -0.7851356 -0.68751552 -0.71222209 -0.69253806 -0.67742852
15 -0.35361572 -0.6394949 -0.67637757 -0.71222209 -0.69253806 -0.61833484
16 -0.49845448 -0.7608621 -0.69494082 -0.71222209 -0.69253806 -0.65280615
17 -0.66459307 -0.7851356 -0.69122817 -0.71222209 -0.69253806 -0.74144668
18 -0.54957405 -0.7183836 -0.69865348 -0.71222209 -0.69253806 -0.50507194
20 -0.79026200 -0.7001785  0.31118768 -0.44206888 -0.69253806 -0.74144668
22 -0.57939379 -0.2814616 -0.44619319 -0.29471259  0.03116421 -0.23422588
23 -0.28332632 -0.3178718 -0.42762993 -0.41750950 -0.31164213 -0.35733773
25 -0.79026200  0.9322106  2.89519300  0.07367815 -0.46400050 -0.72667326
28 -0.72636254 -0.8579559 -0.69865348 -0.71222209 -0.69253806 -0.73652221
29  0.41530774 -0.6819734 -0.69865348 -0.71222209 -0.69253806 -0.49029852
31 -0.75831227 -0.8700926 -0.62440045 -0.68766270 -0.69253806 -0.74144668
33 -0.72210258 -0.4392390 -0.11576722 -0.07367815  0.09464687 -0.09141614
34 -0.39408537 -0.7547938 -0.68380287 -0.71222209 -0.69253806 -0.60848589
35 -0.50484443 -0.7062469 -0.59469924 -0.51574703 -0.66714500 -0.52969431
36 -0.21090693 -0.6212898 -0.67637757 -0.71222209 -0.69253806 -0.56909010
37 -0.36426563 -0.7001785 -0.68380287 -0.71222209 -0.69253806 -0.65280615
38 -0.74553238 -0.7426570 -0.55757273 -0.68766270 -0.65444847 -0.66265510
39 -0.70293274 -0.5848797 -0.49074500 -0.54030641 -0.36242825 -0.50507194
40 -0.38982541  0.3981948 -0.03037624  0.29471259  0.53902546  0.14988307
41 -0.76896218 -0.8154774 -0.66523961 -0.71222209 -0.67984153 -0.71189984
42 -0.70080276 -0.6880418 -0.69494082 -0.71222209 -0.69253806 -0.74144668
43 -0.46224479 -0.7123152 -0.66895227 -0.71222209 -0.69253806 -0.69220194
45 -0.46863474 -0.4695808 -0.52787152 -0.54030641 -0.56557275 -0.51492089
48 -0.79026200 -0.8700926 -0.66523961 -0.54030641 -0.61635888 -0.64295721
50 -0.77322214 -0.8033406 -0.68751552 -0.51574703 -0.55287622 -0.64295721
51 -0.75831227 -0.8215457 -0.62068780 -0.71222209 -0.69253806 -0.67250405
52 -0.72210258 -0.6516316 -0.52044621 -0.31927197 -0.41321438 -0.52969431
53 -0.53679416 -0.8215457 -0.68009022 -0.71222209 -0.69253806 -0.73652221
```

Hide

```
# Mean values of each cluster
aggregate(clustering, by=list(model1$cluster), mean)
```

```
  Group.1   Oct_10    Nov_10    Dec_10    Jan_11    Feb_11    Mar_11
1       1 929.0625 371.87500 470.81250 78.000000 149.06250 413.93750
2       2 142.0769  53.87179  72.23077  8.897436  15.76923  42.51282
```

Hide

```
#different plots to be combined into a plot matrix
ggpairs(cbind(clustering, Cluster=as.factor(model1$cluster)),
        columns=1:6, aes(colour=Cluster, alpha=0.5),
        lower=list(continuous="points"),
        axisLabels="none")
```