Tutorial 10 & 11, Lecture 11

Dawn Cheung

2024-04-28

Tutorial 10 (Topic 8): k-means

the k-means algorithm finds out the distance among each element in your data, then find the number of centroids, allocate the element to the nearest centroids to form clusters, and the ultimate goal is to keep the size of each cluster as small as possible.

- note k is number of clusters / “groups”
- WSS: The sum distance within the centroids
    - small WSS => better model
- STANDARDISE THE INPUTS FOR K-means

```
K = 15
wss <- numeric(K)

for (k in 1:K) {
   wss[k] <- sum(kmeans(scale(data[,c("floor_area_sqm","resale_price")]), centers=k)$withins
s)
}


plot(1:K, wss, col = "blue", type="b", xlab="Number of Clusters",  ylab="Within Sum of Square
s")
```

Suppose we have data for five objects on two features:

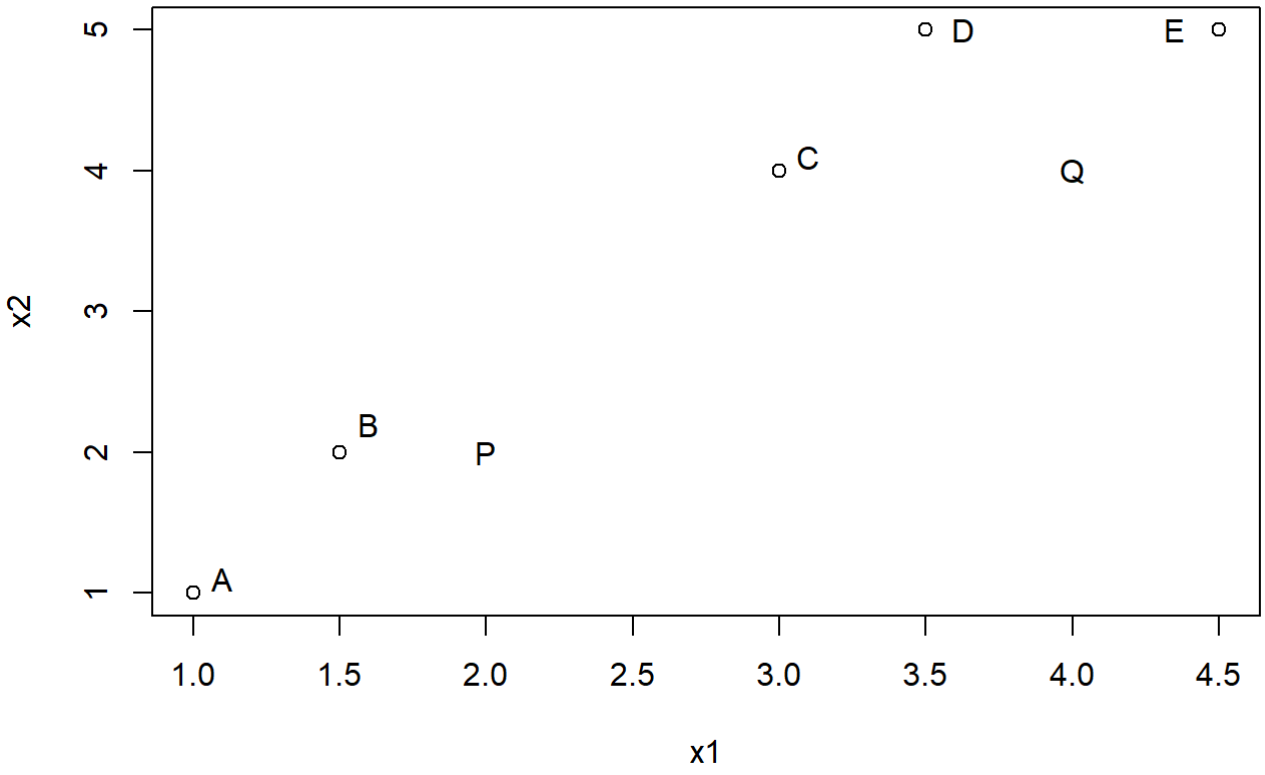
object x1 x2

```
 A  1   1
 B  1.5 2
 C  3   4
 D  3.5 5
 E  4.5 5
```

We set k = 2 to cluster the five data points into two clusters, P and Q, and initialize the algorithm with the centroids (x1,P , x2,P) = (2, 2) and (x1,Q, x2,Q) = (4, 4).

a. Identify the objects in each cluster during the first iteration of the k-means algorithm

```
##not possible to do via kmeans() => use plot() mnaually

x1 = c(1, 1.5, 3, 3.5, 4.5)
x2 = c(1, 2, 4, 5, 5)

objs = cbind(x1, x2)
plot(objs)

text(1.1, 1.1, "A")
text(1.6, 2.2, "B")
text(3.1, 4.1, "C")
text(3.63, 5, "D")
text(4.35, 5, "E")

text(2,2,"P")
text(4,4,"Q")
```




```
###so now you can see quite clearly which points are nearer to which centroids

##tho u can also calculate using Euclidean distance
```

b. Compute the new centroids for the two clusters based on cluster assignment in (a).

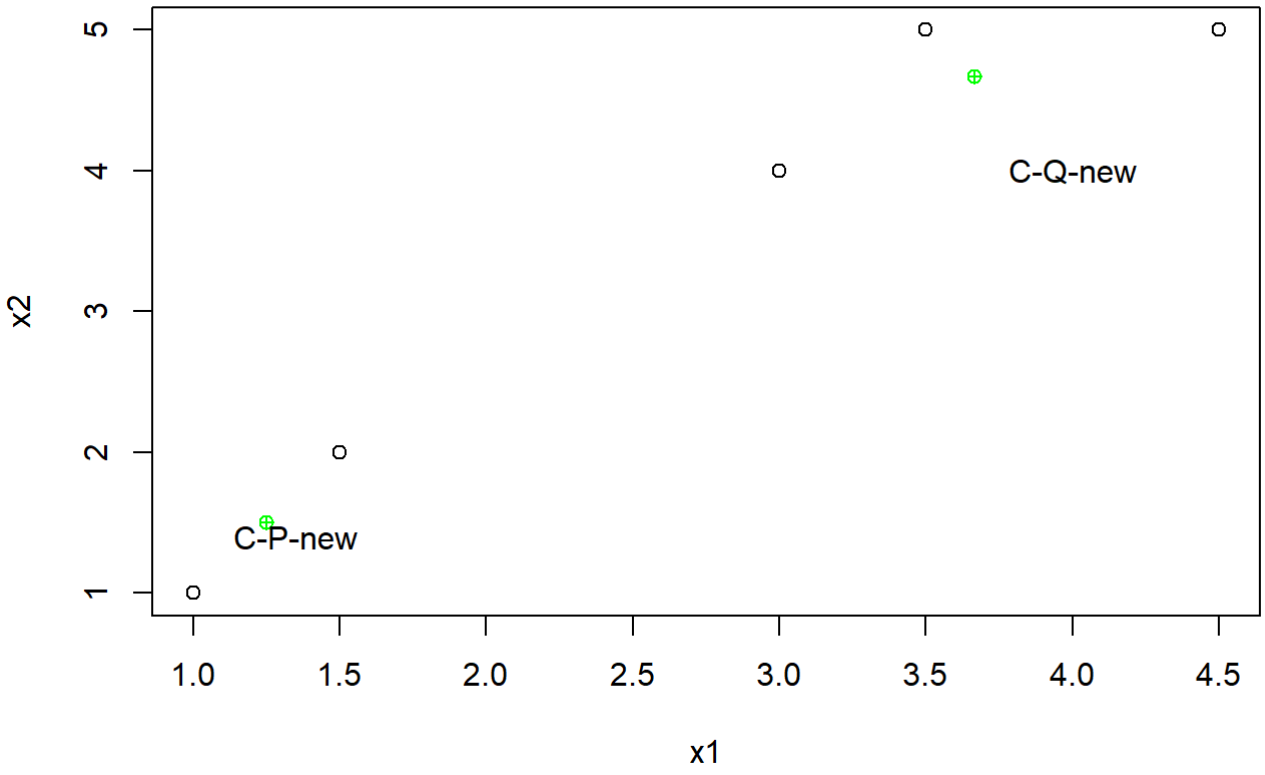
```
#P now has A and B
centriodP = c((1+1.5)/2, (1+2)/2)

#Q now has C, D, E
centriodQ = c((3+3.5+4.5)/3, (4+5+5)/3)

#im more or less literally doing averages btw

#but note that we're finding the MIDPOINT of all the points belonging to the different cluste
rs

#and previous proposed centriods (not part of OG dataset) is removed / not considered

centriodP; centriodQ
```

```
## [1] 1.25 1.50
```

```
## [1] 3.666667 4.666667
```

c. Based on the centroids computed in (b), identify the objects in each cluster during the second iteration of the k-means algorithm.

```
plot(objs)

points(centriodP[1], centriodP[2], pch = 10, col = "green")
points(centriodQ[1], centriodQ[2], pch = 10, col = "green")

text(1.35, 1.4, "C-P-new")
text(4, 4, "C-Q-new")
```



note that since the points identified for the different clusters remains the same, the algorithm is converged! stable!

d. Calculate the Within Sum of Squares (WSS) for the clustering assignment in (c).

```
data = data.frame(x1, x2)
data
```

```
##    x1 x2
## 1 1.0  1
## 2 1.5  2
## 3 3.0  4
## 4 3.5  5
## 5 4.5  5
```

```
kout = kmeans(data, centers = 2)
kout
```

```
## K-means clustering with 2 clusters of sizes 2, 3
##
## Cluster means:
##         x1       x2
## 1 1.250000 1.500000
## 2 3.666667 4.666667
##
## Clustering vector:
## [1] 1 1 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 0.625000 1.833333
##  (between_SS / total_SS =  88.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
kout$withinss
```

```
## [1] 0.625000 1.833333
```

```
kout$tot.withinss #tbh idk what this one is doing
```

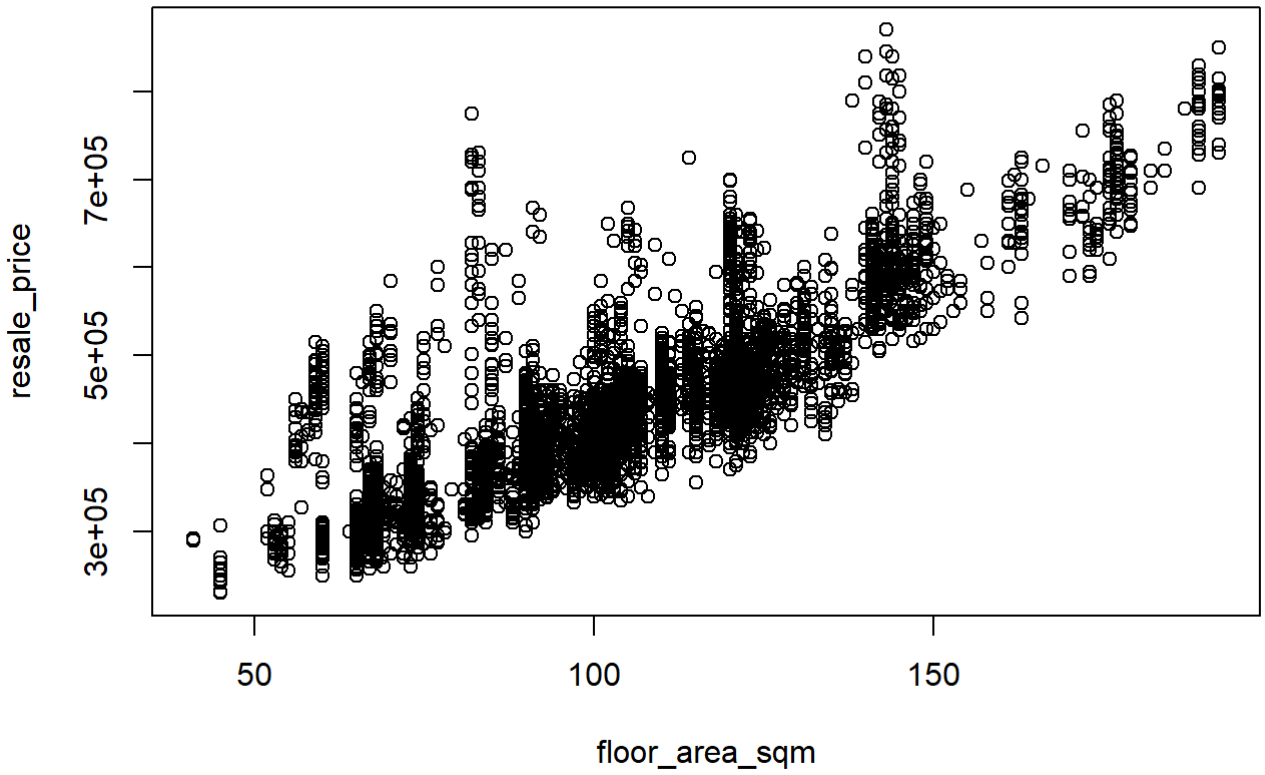
```
## [1] 2.458333
```

(K-Means) Consider data set hdb-2012-to-2014.csv which was extracted from the published data 1

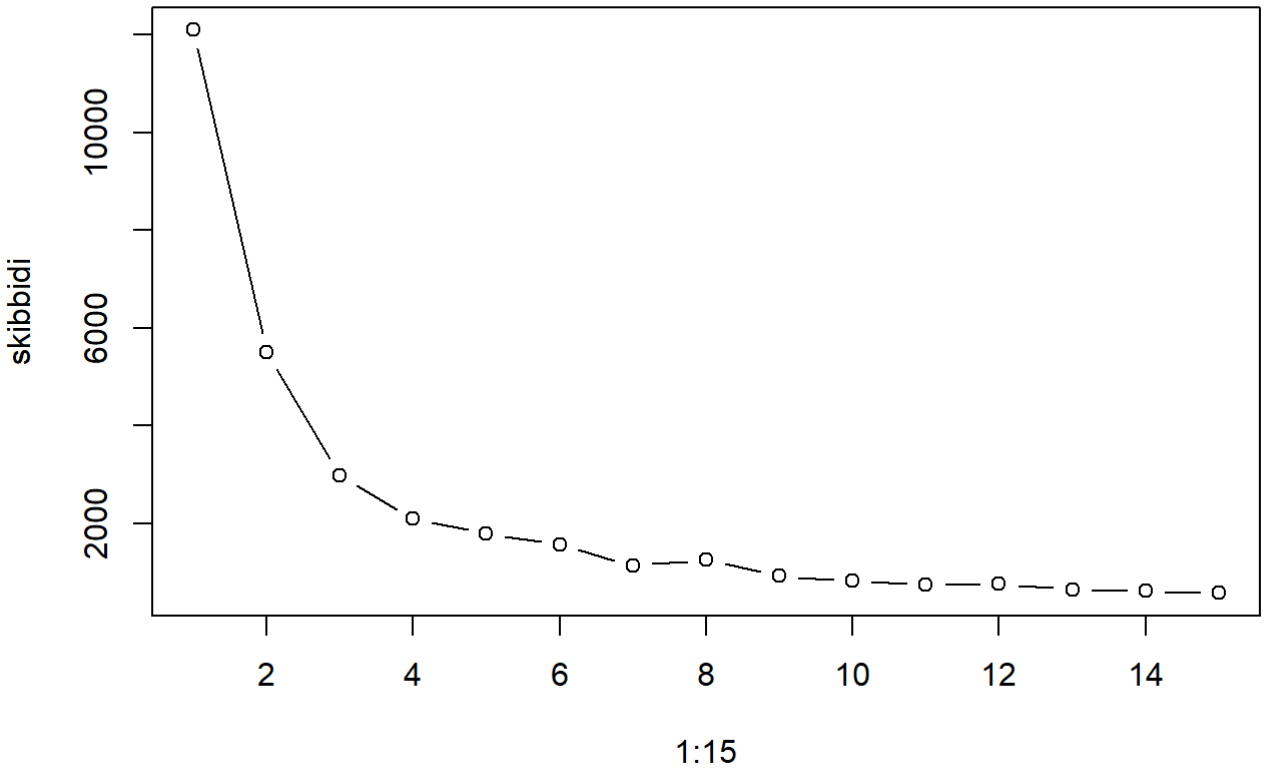
The file has information on the HDB resale flats from Jan 2012 to Dec 2014.

a. Load data into R. Use k means algorithm to pick an optimal value for k in term of WSS, based on two variables, resale_price and floor_area_sqm.

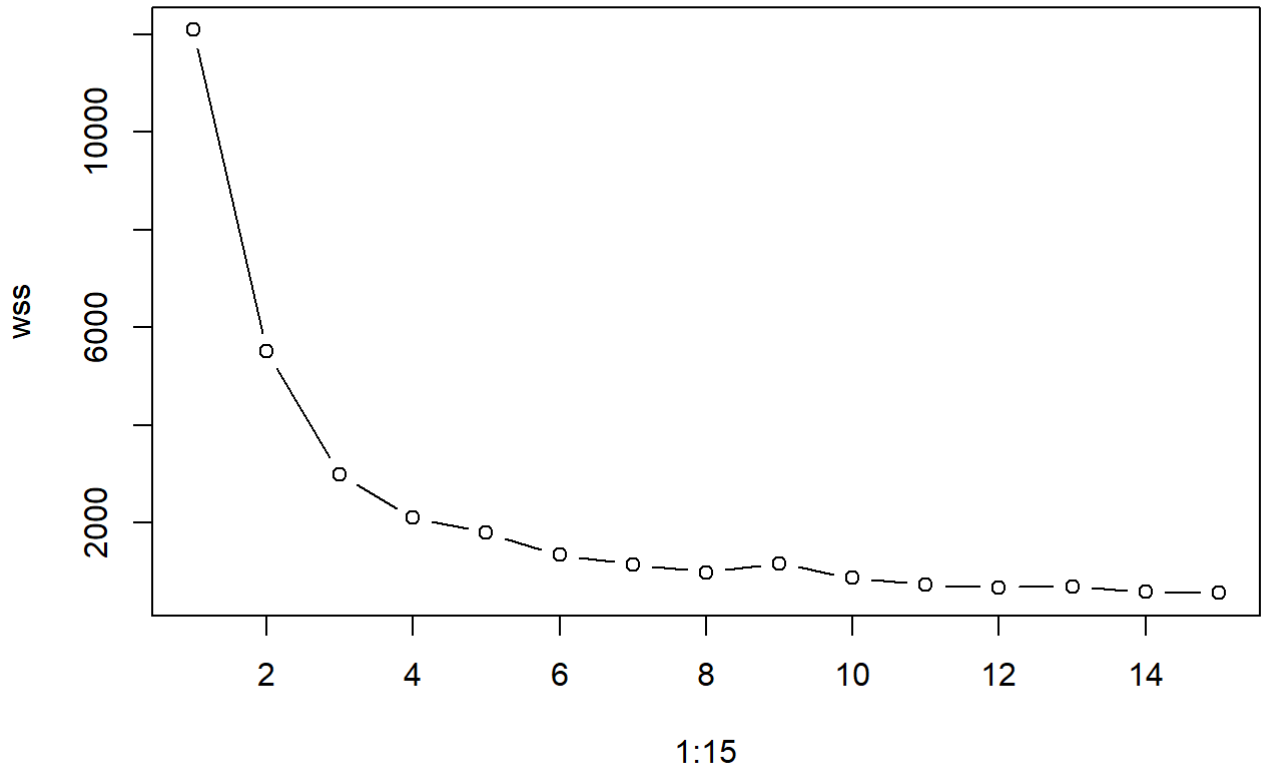
```
sendhelp = read.csv("~/Github/DSA1101 Slayers/datasets/hdb-2012-to-2014.csv")
attach(sendhelp)

plot(floor_area_sqm, resale_price)#so we notice that the resale price is damnn high compared
to the floor_area_sqm => MUST SCALE FEATURES
```




```
skibbidi = c() #putting all the wss here

for (i in 1:15) {
  skibbidi[i] = sum(kmeans(scale(sendhelp[,c("floor_area_sqm","resale_price")]), centers=i)$w
ithinss)
}
plot(1:15, skibbidi, type = "b")
```

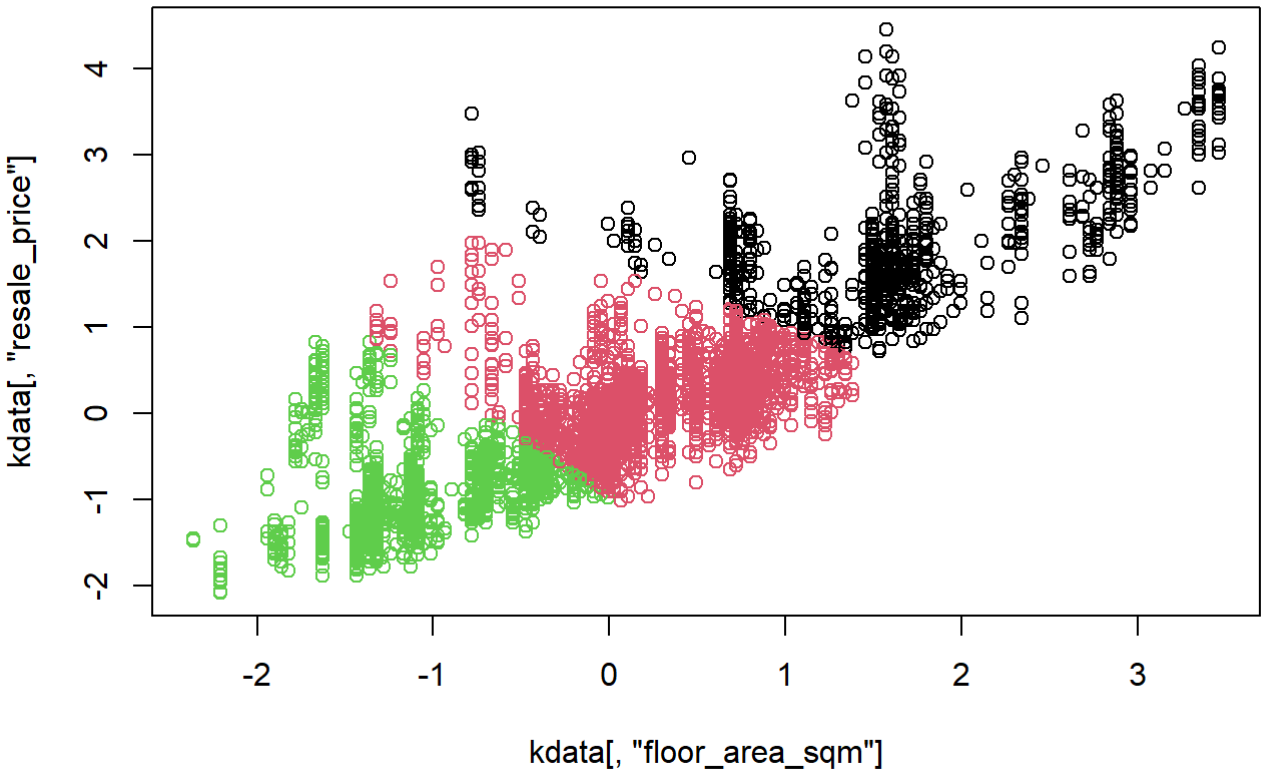



```
kdata = scale(sendhelp[, c("floor_area_sqm","resale_price")])
wss = numeric(15)

for (i in 1:15) {
  wss[i] = sum(kmeans(kdata, centers = i)$withinss)
}

plot(1:15, wss, type = "b")
```



b. With the optimal k in part (a), plot the data points in the k clusters determined.

```
kout = kmeans(kdata, centers = 3)

plot(kdata[, "floor_area_sqm"], kdata[, "resale_price"], col = kout$cluster)
```



Tutorial 11 (Topic 9): association rules

A local retailer has a database that stores 10,000 transactions of last summer. After analyzing the data, a data science team has identified the following statistics:

- {battery} appears in 6000 transactions
- {sunscreen} appears in 5000 transactions
- {sandals} appears in 4000 transactions
- {bowls} appears in 2000 transactions
- {battery, sunscreen} appears in 1500 transactions
- {battery, sandals} appears in 1000 transactions
- {battery, bowls} appears in 250 transactions
- {battery, sunscreen, sandals} appears in 600 transactions

a. What are the support values of the preceding itemsets?

```
sigh = c()

batsup = 6000 / 10000
sunsup = 5000 / 10000
sansup = 4000 / 10000
bowsup = 2000 / 10000
bat_sun_sup = 1500 / 10000
bat_san_sup = 1000 / 10000
bat_bow_sup = 250 / 10000
bat_sun_san_sup = 600 / 10000

sigh = c(batsup, sunsup, sansup, bowsup, bat_sun_sup, bat_san_sup, bat_bow_sup, bat_sun_san_s
up)
```

b. Assuming the minimum support is 0.05, which itemsets are considered frequent?

```
sigh[which(sigh > 0.05)] #then manually list the itemsets idk
```

```
## [1] 0.60 0.50 0.40 0.20 0.15 0.10 0.06
```

c. What are the confidence values of {battery} → {sunscreen} and {battery, sunscreen} → {sandals}? Which of these two rules is more interesting, i.e. has higher values of confidence?

2. Suppose for three products A, B and C, support({A}) = 0.6, support({B}) = 0.6, con□dence({B} → {A}) = 0.9 and confidence({C} → {A, B}) = 0.5. Compute the following quantities.

a. Lift({A} → {B})
b. Leverage({A} → {B})
c. Confidence({A} → {B})
d. Lift({A, B} → {C})

Lecture 11 (Topic 9)

```
library("arules")
```

```
## Warning: package 'arules' was built under R version 4.3.3
```

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

```
## Warning: package 'Matrix' was built under R version 4.3.2
```

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

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

```
library("arulesViz")
```

```
## Warning: package 'arulesViz' was built under R version 4.3.3
```

```
data(Groceries) #alr installed in R library arules; everyone can access it

#sparse matrix: specifically for association rules, stored in a dot and slash format
summary(Groceries)
```

```
## transactions as itemMatrix in sparse format with
##  9835 rows (elements/itemsets/transactions) and
##  169 columns (items) and a density of 0.02609146
##
## most frequent items:
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   55   46
##   17   18   19   20   21   22   23   24   26   27   28   29   32
##   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##        labels  level2           level1
## 1 frankfurter sausage meat and sausage
## 2     sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

```
inspect(head(Groceries)) #inspect is under library arules
```

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
## [4] {pip fruit,
##      yogurt,
##      cream cheese ,
##      meat spreads}
## [5] {other vegetables,
##      whole milk,
##      condensed milk,
##      long life bakery product}
## [6] {whole milk,
##      butter,
##      yogurt,
##      rice,
##      abrasive cleaner}
```

```
inspect(head(Groceries, 10)) #inspect first 10 itemsets instead of first 6
```

```
##      items
## [1]  {citrus fruit,
##       semi-finished bread,
##       margarine,
##       ready soups}
## [2]  {tropical fruit,
##       yogurt,
##       coffee}
## [3]  {whole milk}
## [4]  {pip fruit,
##       yogurt,
##       cream cheese ,
##       meat spreads}
## [5]  {other vegetables,
##       whole milk,
##       condensed milk,
##       long life bakery product}
## [6]  {whole milk,
##       butter,
##       yogurt,
##       rice,
##       abrasive cleaner}
## [7]  {rolls/buns}
## [8]  {other vegetables,
##       UHT-milk,
##       rolls/buns,
##       bottled beer,
##       liquor (appetizer)}
## [9]  {pot plants}
## [10] {whole milk,
##       cereals}
```

```
Groceries@itemInfo[1:10,] #all columns
```

```
##               labels  level2           level1
## 1        frankfurter sausage meat and sausage
## 2            sausage sausage meat and sausage
## 3         liver loaf sausage meat and sausage
## 4                ham sausage meat and sausage
## 5               meat sausage meat and sausage
## 6  finished products sausage meat and sausage
## 7    organic sausage sausage meat and sausage
## 8            chicken poultry meat and sausage
## 9             turkey poultry meat and sausage
## 10              pork    pork meat and sausage
```

```
Groceries@data[,100:105] #all rows
```

```
## 169 x 6 sparse Matrix of class "ngCMatrix"
##
##   [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,] . . . . . .
##  [48,] . . . . . .
##  [49,] . . . . . .
##  [50,] . . . . . .
##  [51,] . . . . . .
##  [52,] . . . . . .
##  [53,] . . . . . .
```

```
##  [54,] . . . . . .
##  [55,] . . . . . .
##  [56,] . . | | . .
##  [57,] . . . . . .
##  [58,] . . . . . .
##  [59,] . . | . . .
##  [60,] . . . . . .
##  [61,] . . . . . .
##  [62,] . . . . . .
##  [63,] . . . . . .
##  [64,] . . . . . .
##  [65,] . . . . . .
##  [66,] . . . . . .
##  [67,] . . . . . .
##  [68,] . . . . . .
##  [69,] . . . . . .
##  [70,] . . . . . .
##  [71,] . . . . . .
##  [72,] . . . . . .
##  [73,] . . . . . .
##  [74,] . . . . . .
##  [75,] . . . . . .
##  [76,] . . . . . .
##  [77,] . . . . . .
##  [78,] . . . . . .
##  [79,] . . . . . .
##  [80,] . . . . . .
##  [81,] . . . . . .
##  [82,] . . . . . .
##  [83,] . . . . . .
##  [84,] . . . . . .
##  [85,] . . . . . .
##  [86,] . . . . . .
##  [87,] . . . . . .
##  [88,] . . . . . .
##  [89,] . . . . . .
##  [90,] . . . . . .
##  [91,] . . . . . .
##  [92,] . . . . . .
##  [93,] . . . . . .
##  [94,] . . . . . .
##  [95,] . . . . . .
##  [96,] . . . . . .
##  [97,] . . . . . .
##  [98,] . . . . . .
##  [99,] . . . . | .
## [100,] . . . . . .
## [101,] . . . . . .
## [102,] . . . . . .
## [103,] . . . | . .
## [104,] . | | . . |
## [105,] . | . . . .
## [106,] . . . . | .
## [107,] . . . . . .
## [108,] . . . . | .
## [109,] . . . . . .
```

```
## [110,] . . . . . .
## [111,] . . . . . .
## [112,] . . . . . .
## [113,] . . . . . .
## [114,] . . . . . .
## [115,] . . . . . .
## [116,] . . . . . .
## [117,] . . . . . .
## [118,] . . . . . .
## [119,] . . . . . .
## [120,] . . . . . .
## [121,] . . . . . .
## [122,] . . . . . .
## [123,] . . . . . .
## [124,] . . . . . .
## [125,] . . . . . .
## [126,] . . . . . .
## [127,] . . . . . .
## [128,] . . . . . .
## [129,] . . . . . .
## [130,] . . . . . .
## [131,] . . | . . .
## [132,] . . . . . .
## [133,] . . . . . .
## [134,] . . . . . .
## [135,] . . . . . .
## [136,] . . . . | .
## [137,] . . . . . .
## [138,] . . . . . .
## [139,] . . . . . .
## [140,] . . . . . .
## [141,] . . . . . .
## [142,] . . | . . .
## [143,] . . . . . .
## [144,] . . . . . .
## [145,] . . . . . .
## [146,] . . . . . .
## [147,] . . . . . .
## [148,] . . . . . .
## [149,] . . . . . .
## [150,] . . . . . .
## [151,] . . . . . .
## [152,] . . . . . .
## [153,] . . . . | .
## [154,] . . . . . .
## [155,] . . . . . .
## [156,] . . . . . .
## [157,] . . . . . .
## [158,] . . . . . .
## [159,] . . . . . .
## [160,] . . . . . .
## [161,] . . . . . .
## [162,] . . . . . .
## [163,] . . . . . .
## [164,] . . . . | .
## [165,] . . . . . .
```

```
## [166,] . . . . . .
## [167,] . . . . . .
## [168,] . . . . | .
## [169,] . . . . . .
```

```
#sparse matrix: specifically for association rules, stored in a dot and slash format

#see the items for the first 5 transactions
apply(Groceries@data[,1:5], 2,
      function(r) paste(Groceries@itemInfo[r, "labels"], collapse = ", "))
```

```
## [1] "citrus fruit, semi-finished bread, margarine, ready soups"
## [2] "tropical fruit, yogurt, coffee"
## [3] "whole milk"
## [4] "pip fruit, yogurt, cream cheese , meat spreads"
## [5] "other vegetables, whole milk, condensed milk, long life bakery product"
```

```
#2: 2nd column
#so this converts the sparse matrix into a readable itemset

#see the items for 100 to 105th transactions (index)
apply(Groceries@data[,100:105], 2,
      function(r) paste(Groceries@itemInfo[r, "labels"], collapse = ", "))
```

```
## [1] "citrus fruit, tropical fruit"
## [2] "soda, misc. beverages"
## [3] "sausage, pork, grapes, whole milk, rolls/buns, pastry, soda, specialty bar, bathroom
cleaner"
## [4] "frankfurter, rolls/buns, bottled water"
## [5] "sausage, whole milk, yogurt, coffee, fruit/vegetable juice, bottled beer, softener, n
apkins, photo/film, shopping bags"
## [6] "soda"
```

```
#get the frequent 1-itemset



itemset.1 = apriori(Groceries, parameter = list(minlen = 1, maxlen = 1, support = 0.04, targe
t = "frequent itemsets"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##          NA    0.1    1 none FALSE            TRUE       5    0.04      1
##  maxlen            target  ext
##       1 frequent itemsets TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 393
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [32 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1
```

```
## Warning in apriori(Groceries, parameter = list(minlen = 1, maxlen = 1, support
## = 0.04, : Mining stopped (maxlen reached). Only patterns up to a length of 1
## returned!
```

```
##  done [0.00s].
## sorting transactions ... done [0.00s].
## writing ... [32 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(itemset.1)
```

```
## set of 32 itemsets
##
## most frequent items:
## frankfurter     sausage     chicken        pork        beef     (Other)
##           1           1           1           1           1          27
##
## element (itemset/transaction) length distribution:sizes
##  1
## 32
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1       1       1       1       1       1
##
## summary of quality measures:
##     support            count
##  Min.   :0.04209   Min.   : 414.0
##  1st Qu.:0.05709   1st Qu.: 561.5
##  Median :0.07397   Median : 727.5
##  Mean   :0.08909   Mean   : 876.2
##  3rd Qu.:0.10013   3rd Qu.: 984.8
##  Max.   :0.25552   Max.   :2513.0
##
## includes transaction ID lists: FALSE
##
## mining info:
##       data ntransactions support confidence
##  Groceries          9835    0.04          1
##
call
##  apriori(data = Groceries, parameter = list(minlen = 1, maxlen = 1, support = 0.04, target
= "frequent itemsets"))
```

```
inspect(sort(itemset.1, by = "support"))
```

```
##      items                   support    count
## [1]  {whole milk}            0.25551601 2513
## [2]  {other vegetables}      0.19349263 1903
## [3]  {rolls/buns}            0.18393493 1809
## [4]  {soda}                  0.17437722 1715
## [5]  {yogurt}                0.13950178 1372
## [6]  {bottled water}         0.11052364 1087
## [7]  {root vegetables}       0.10899847 1072
## [8]  {tropical fruit}        0.10493137 1032
## [9]  {shopping bags}         0.09852567  969
## [10] {sausage}               0.09395018  924
## [11] {pastry}                0.08896797  875
## [12] {citrus fruit}          0.08276563  814
## [13] {bottled beer}          0.08052872  792
## [14] {newspapers}            0.07981698  785
## [15] {canned beer}           0.07768175  764
## [16] {pip fruit}             0.07564820  744
## [17] {fruit/vegetable juice} 0.07229283  711
## [18] {whipped/sour cream}    0.07168277  705
## [19] {brown bread}           0.06487036  638
## [20] {domestic eggs}         0.06344687  624
## [21] {frankfurter}           0.05897306  580
## [22] {margarine}             0.05856634  576
## [23] {coffee}                0.05805796  571
## [24] {pork}                  0.05765125  567
## [25] {butter}                0.05541434  545
## [26] {curd}                  0.05327911  524
## [27] {beef}                  0.05246568  516
## [28] {napkins}               0.05236401  515
## [29] {chocolate}             0.04961871  488
## [30] {frozen vegetables}     0.04809354  473
## [31] {chicken}               0.04290798  422
## [32] {white bread}           0.04209456  414
```

```
itemset.2 = apriori(Groceries, parameter = list(minlen = 2, maxlen = 2, support = 0.02, targe
t = "frequent itemsets"))
```

```
## 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
##       2 frequent itemsets TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 196
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [59 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2
```

```
## Warning in apriori(Groceries, parameter = list(minlen = 2, maxlen = 2, support
## = 0.02, : Mining stopped (maxlen reached). Only patterns up to a length of 2
## returned!
```

```
##  done [0.00s].
## sorting transactions ... done [0.00s].
## writing ... [61 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(itemset.2)
```

```
## set of 61 itemsets
##
## most frequent items:
##       whole milk other vegetables           yogurt       rolls/buns
##               25               17                9                9
##             soda          (Other)
##                9               53
##
## element (itemset/transaction) length distribution:sizes
##  2
## 61
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       2       2       2       2       2
##
## summary of quality measures:
##     support            count
##  Min.   :0.02003   Min.   :197.0
##  1st Qu.:0.02227   1st Qu.:219.0
##  Median :0.02613   Median :257.0
##  Mean   :0.02951   Mean   :290.3
##  3rd Qu.:0.03223   3rd Qu.:317.0
##  Max.   :0.07483   Max.   :736.0
##
## includes transaction ID lists: FALSE
##
## mining info:
##       data ntransactions support confidence
##  Groceries          9835    0.02          1
##
call
##  apriori(data = Groceries, parameter = list(minlen = 2, maxlen = 2, support = 0.02, target
= "frequent itemsets"))
```

```
inspect(sort(itemset.2, by = "support", 10)) #limit to the first 10 itemsets
```

```
##      items                                     support    count
## [1]  {other vegetables, whole milk}            0.07483477 736
## [2]  {whole milk, rolls/buns}                  0.05663447 557
## [3]  {whole milk, yogurt}                      0.05602440 551
## [4]  {root vegetables, whole milk}             0.04890696 481
## [5]  {root vegetables, other vegetables}       0.04738180 466
## [6]  {other vegetables, yogurt}                0.04341637 427
## [7]  {other vegetables, rolls/buns}            0.04260295 419
## [8]  {tropical fruit, whole milk}              0.04229792 416
## [9]  {whole milk, soda}                        0.04006101 394
## [10] {rolls/buns, soda}                        0.03833249 377
## [11] {tropical fruit, other vegetables}        0.03589222 353
## [12] {whole milk, bottled water}               0.03436706 338
## [13] {yogurt, rolls/buns}                      0.03436706 338
## [14] {whole milk, pastry}                      0.03324860 327
## [15] {other vegetables, soda}                  0.03274021 322
## [16] {whole milk, whipped/sour cream}          0.03223183 317
## [17] {sausage, rolls/buns}                     0.03060498 301
## [18] {citrus fruit, whole milk}                0.03050330 300
## [19] {pip fruit, whole milk}                   0.03009659 296
## [20] {whole milk, domestic eggs}               0.02999492 295
## [21] {sausage, whole milk}                     0.02989324 294
## [22] {tropical fruit, yogurt}                  0.02928317 288
## [23] {bottled water, soda}                     0.02897814 285
## [24] {other vegetables, whipped/sour cream}    0.02887646 284
## [25] {citrus fruit, other vegetables}          0.02887646 284
## [26] {whole milk, butter}                      0.02755465 271
## [27] {whole milk, newspapers}                  0.02735130 269
## [28] {yogurt, soda}                            0.02735130 269
## [29] {sausage, other vegetables}               0.02694459 265
## [30] {whole milk, fruit/vegetable juice}       0.02663955 262
## [31] {whole milk, curd}                        0.02613116 257
## [32] {pip fruit, other vegetables}             0.02613116 257
## [33] {root vegetables, yogurt}                 0.02582613 254
## [34] {whole milk, brown bread}                 0.02521607 248
## [35] {other vegetables, bottled water}         0.02480935 244
## [36] {soda, shopping bags}                     0.02460600 242
## [37] {tropical fruit, rolls/buns}              0.02460600 242
## [38] {whole milk, shopping bags}               0.02450432 241
## [39] {sausage, soda}                           0.02430097 239
## [40] {root vegetables, rolls/buns}             0.02430097 239
## [41] {whole milk, margarine}                   0.02419929 238
## [42] {rolls/buns, bottled water}               0.02419929 238
## [43] {other vegetables, shopping bags}         0.02318251 228
## [44] {yogurt, bottled water}                   0.02297916 226
## [45] {other vegetables, pastry}                0.02257245 222
## [46] {other vegetables, domestic eggs}         0.02226741 219
## [47] {pork, whole milk}                        0.02216573 218
## [48] {pork, other vegetables}                  0.02165735 213
## [49] {citrus fruit, yogurt}                    0.02165735 213
## [50] {beef, whole milk}                        0.02125064 209
## [51] {other vegetables, fruit/vegetable juice} 0.02104728 207
## [52] {pastry, soda}                            0.02104728 207
## [53] {tropical fruit, root vegetables}         0.02104728 207
## [54] {rolls/buns, pastry}                      0.02094560 206
```

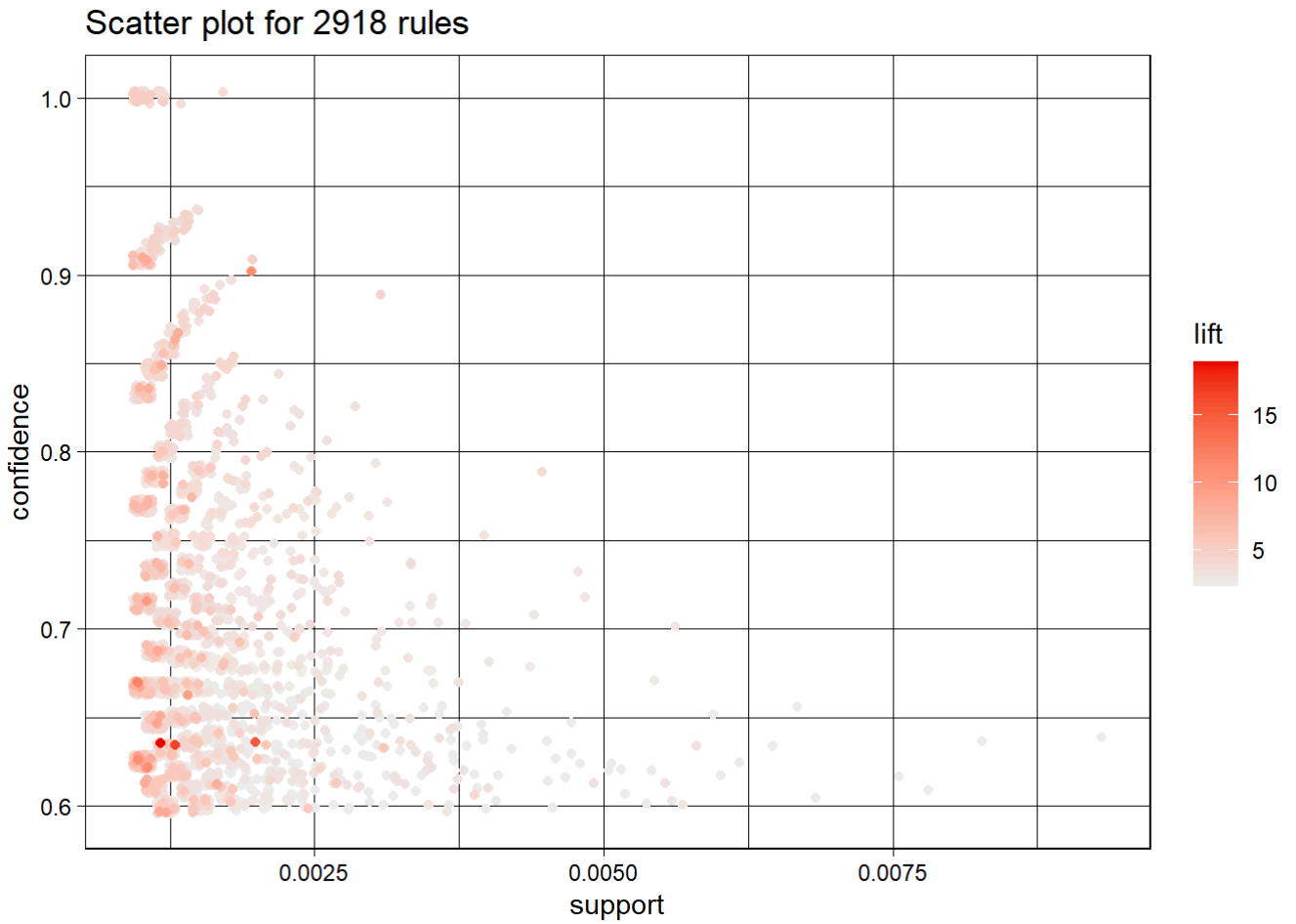
```
## [55] {tropical fruit, soda}                    0.02084392 205
## [56] {yogurt, whipped/sour cream}              0.02074225 204
## [57] {frankfurter, whole milk}                 0.02053889 202
## [58] {whole milk, frozen vegetables}           0.02043721 201
## [59] {whole milk, bottled beer}                0.02043721 201
## [60] {tropical fruit, pip fruit}               0.02043721 201
## [61] {other vegetables, butter}                0.02003050 197
```

```
rules = apriori(Groceries, parameter = list(support = 0.001, confidence = 0.6, target = "rule
s"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [2918 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

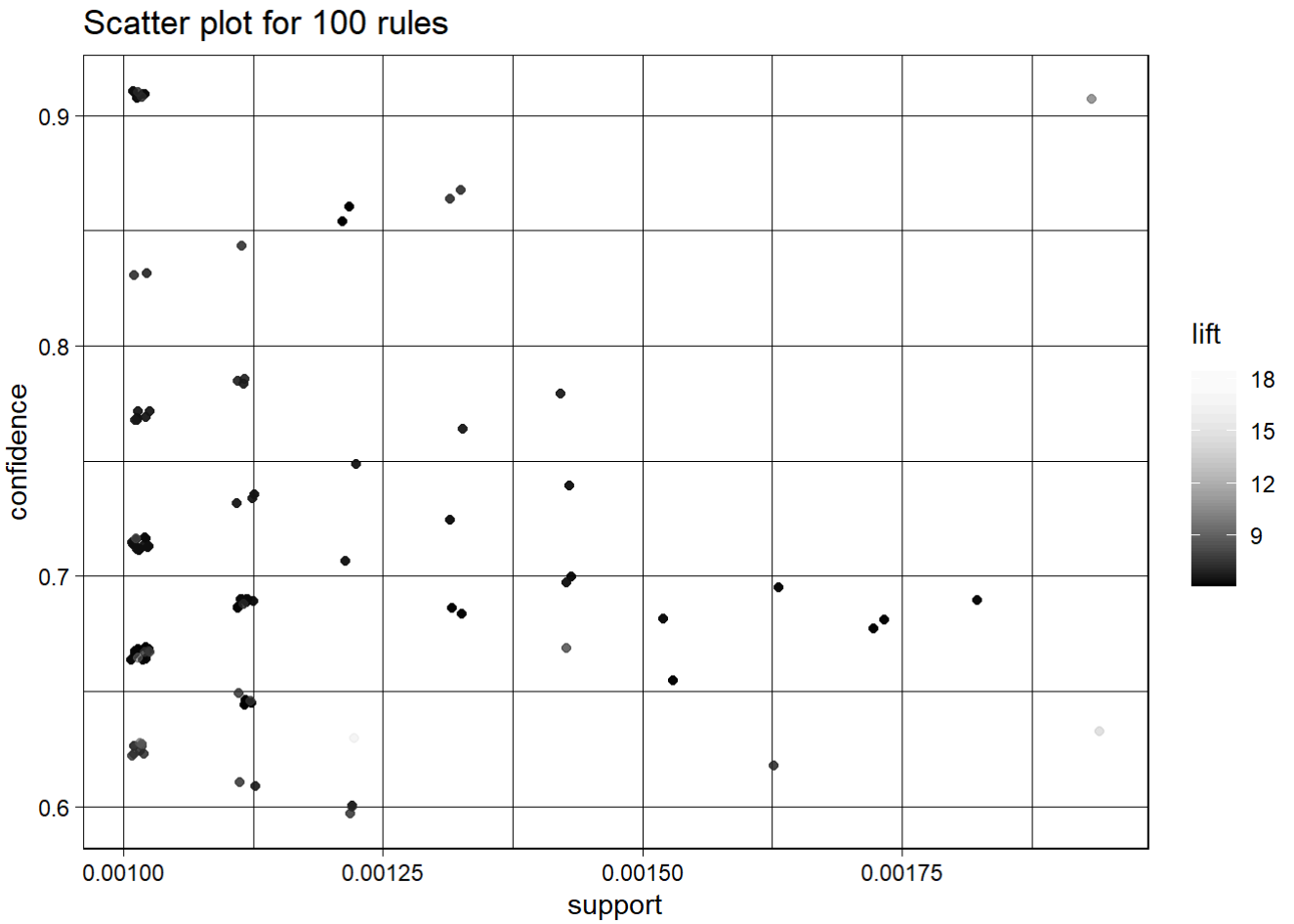
```
plot(rules)
```

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




```
#limit to only 100 dots / rules with the highest lift
plot(rules, measure = c("support", "confidence"), shading = "lift", col = "black", limit = 10
0)
```

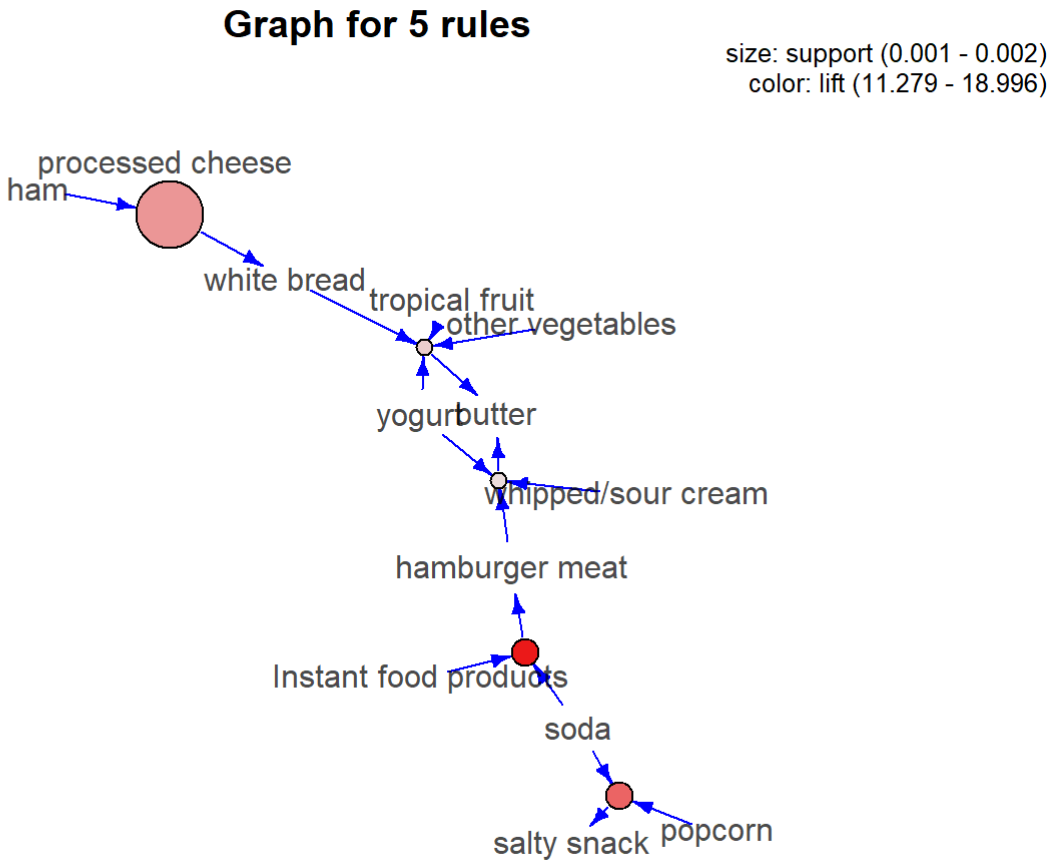
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```




```
#attain the top few rules
inspect(head(sort(rules, by = "lift"), 3))
```

```
##     lhs                              rhs              support     confidence
## [1] {Instant food products, soda} => {hamburger meat} 0.001220132 0.6315789
## [2] {soda, popcorn}               => {salty snack}    0.001220132 0.6315789
## [3] {ham, processed cheese}       => {white bread}    0.001931876 0.6333333
##     coverage    lift     count
## [1] 0.001931876 18.99565 12
## [2] 0.001931876 16.69779 12
## [3] 0.003050330 15.04549 19
```

```
#note that 3 is a parameter for head() ie head() controls how many outputs come out

#plot the top 5 rules for visualisation
highLiftRules = head(sort(rules, by = "lift"), 5)
plot(highLiftRules, method = "graph", engine = "igraph", edgeCol = "blue", alpha = 1)
```




```
# alpha = c(0,1)
# the size of the node is sprted by te support
# the darkness of the node's color represents the change in lift
```