Data Manipulation, Association Rule Mining, and Clustering Analysis

Blake Pappas

2023-12-17

Part I. Understand Basic Concepts in Association Rule Mining

Question 1: After mining some shopping basket dataset, you find that rule "{butter} $\rightarrow$ {bread}" has confidence of 70% and support of 5%. If "butter" appears in 200 shopping baskets (i.e., transactions), what is the size of the entire dataset that was mined? Explain.

Answer: The size of the entire data set that was mined is 2,800.

Explanation:

Confidence = supp(X $\rightarrow$ Y) / supp(X)

Confidence = 70% (0.7)

Butter appears in 200 shopping baskets

Supp(X $\rightarrow$ Y) / 200 = 0.7

Supp(X $\rightarrow$ Y) = 140 <- Butter and bread appearing in the same shopping baskets

Support = # transactions containing both X and Y / # transactions

Support = 5% (0.05)

140 / # transactions = 0.05; # transactions = 2,800

Question 2: Consider two association rules mined from some shopping basket dataset: "{cereal} $\rightarrow$ {milk}" and "{cereal, cookies} $\rightarrow$ {milk}". Which of the following is true about the support of these rules? Explain.

a. Support({cereal}$\rightarrow${milk}) $\geq$ Support({cereal, cookies}$\rightarrow${milk})
b. Support({cereal}$\rightarrow${milk}) $\leq$ Support({cereal, cookies}$\rightarrow${milk})
c. Not enough information, i.e., it could be either $\geq$ or $\leq$, depending on the specific dataset.

Answer: a. Support({cereal} → {milk}) $\geq$ Support({cereal, cookies} → {milk})

Explanation:

The fraction of transactions containing cereal and milk has to be greater than the fraction of transactions containing cereal, cookies, and milk.

As an example:

4 customers visited a coffee shop one morning, generating a dataset of 4 transactions:

1. {yogurt, cereal, milk}
2. {milk, cereal, cookies}
3. {yogurt, milk, cereal, cookies}
4. {milk, cookies}

Support({cereal} → {milk}) = # transactions containing both cereal and milk / # transactions

3 / 4 = 75%

Support({cereal} → {milk}) = 75%

Support({cereal, cookies} → {milk}) = # transactions containing cereal, cookies, and milk / # transactions

2 / 4 = 50%

Support({cereal, cookies} → {milk}) = 50%

Support({cereal} → {milk}) = 75% $\geq$ Support({cereal, cookies} → {milk}) = 50%

Question 3: Consider two association rules mined from some shopping basket dataset: "{cereal} → {milk}" and "{cereal} → {milk, yogurt}". Which of the following is true about the confidence of these rules? Explain.

a. Confidence({cereal} → {milk}) $\geq$ Confidence({cereal} → {milk, yogurt})
b. Confidence({cereal} → {milk}) $\leq$ Confidence({cereal} → {milk, yogurt})
c. Not enough information, i.e., it could be either $\geq$ or $\leq$, depending on the specific dataset.

Answer: a. Confidence({cereal} → {milk}) $\geq$ Confidence({cereal} → {milk, yogurt})

Explanation:

The probability of milk appearing in transactions that contain cereal must be greater than the probability of milk and yogurt appearing in the same set of transactions containing cereal.

As an example:

4 customers visited a coffee shop one morning, generating a dataset of 4 transactions:

1. {yogurt, cereal, milk}
2. {milk, cereal, cookies}
3. {yogurt, milk, cereal, cookies}

4. {milk, cookies}

Confidence({cereal} → {milk}) = Supp({cereal} → {milk}) / Supp({cereal})

3 / 3 = 100%

Confidence({cereal} → {milk}) = 100%

Confidence({cereal} → {milk, yogurt}) = Supp({cereal} → {milk, yogurt}) / Supp({cereal})

2 / 3 = 67%

Confidence({cereal} → {milk, yogurt}) = 67%

Confidence({cereal} → {milk}) (100%) $\geq$ Confidence({cereal} → {milk, yogurt}) (67%)

Part II. Understand Basic Concepts in Clustering Analysis

Question 4: Consider the following "exam.csv" dataset, which shows the results of 3 exams taken by 4 different students (all scores are on the 0-100 scale). Find which two students are most similar to each other (i.e., have smallest distance between them) according to the following distance metrics: **(i) Euclidean, (ii) Manhattan, (iii) max-coordinate**.

```r
library(dplyr)

library(stats)

exam = read.csv("exam.csv")
```

```r
# Euclidean
euclidian_distance_matrix = dist(exam[, 2:4], method = "euclidean")

as.matrix(euclidian_distance_matrix)
```

```
##          1        2        3        4
## 1  0.00000 48.06246 81.36953 61.35145
## 2 48.06246  0.00000 51.70106 65.83312
## 3 81.36953 51.70106  0.00000 49.16299
## 4 61.35145 65.83312 49.16299  0.00000
```

```r
# Manhattan
manhattan_distance_matrix = dist(exam[, 2:4], method = "manhattan")

as.matrix(manhattan_distance_matrix)
```

```
##     1   2   3   4
## 1   0  64 119  86
## 2  64   0  81 108
## 3 119  81   0  63
## 4  86 108  63   0
```

```r
# Max-Coordinate
max_distance_matrix = dist(exam[, 2:4], method = "maximum")

as.matrix(max_distance_matrix)
```

```
##    1  2  3  4
## 1  0 46 64 58
## 2 46  0 45 53
## 3 64 45  0 48
## 4 58 53 48  0
```

Answers:

Euclidian: K and L are most similar to each other.

Manhattan: M and N are most similar to each other.

Max-Coordinate: L and M are most similar to each other.

Question 5: Consider the following three item lists, which reflect purchases from a small grocery store. Calculate the **Jaccard distance** between each pair of baskets – i.e., calculate $J(A, B)$, $J(B, C)$ and $J(A, C)$. Note: You may find the distances easier to compute if you first convert the item lists into a binary matrix format.

- Basket A = {Pudding, Jam, Salsa}
- Basket B = {Salsa, Pudding, Chips}
- Basket C = {Jam, Chips, Salsa, Pudding}

Answer the following two sub-questions:

Question 5.1: Which two baskets are farthest apart based on Jaccard distance?

Jaccard Distance : $d(A, B) = \mathrm{N}_{01} + \mathrm{N}_{10} \ / \ \mathrm{N}_{01} + \mathrm{N}_{10} + \mathrm{N}_{11}$

$J(A, B) = 2/4 \rightarrow 1/2$

$J(B, C) = 1/4$

$J(A, C) = 1/4$

Answer: Baskets A and B are farthest apart, based on Jaccard distance.

Question 5.2: Assume that this small grocery store ***only*** sells 4 products: {Pudding, Jam, Salsa, Chips}. Bearing in mind that the Jaccard distance is well suited to asymmetric binary data, is it a good idea to use it in the context described here? Why or why not?

Answer: Yes, it is a good idea to use Jaccard distance in the context described here. Given that there are only four items sold at this grocery store, N_{00} is not as important as N_{11}. In other words, two people buying the same product is more informative to management than neither person buying the same product. In fact, the three item list provided includes no instance of N_{00}.

Question 6: Consider the following "customer.csv" dataset.

Answer the following two sub-questions:

Question 6.1: Use the **Euclidean** distance metric *directly* on the given data to determine which customer (among B, C, and D) is most similar to customer A.

```
customer = read.csv("customer.csv")

euclidian_distance_matrix = dist(customer[,2:4], method = "euclidean")

as.matrix(euclidian_distance_matrix)
```

```
##          1        2        3        4
## 1    0.000 6000.053 9500.003 1500.091
## 2 6000.053    0.000 3500.046 4500.009
## 3 9500.003 3500.046    0.000 8000.005
## 4 1500.091 4500.009 8000.005    0.000
```

Answer: Based on the Euclidian distance metric directly on the given data, customer D is most similar to customer A.

Question 6.2: **Normalize** the customer data using **min-max normalization**. Then, use the Euclidean distance metric on the normalized data to determine which customer (among B, C, and D) is most similar to customer A.

```
normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
  }

customer_normalized = customer %>%
  mutate_at(c(2:4), normalize)

euclidian_distance_matrix = dist(customer_normalized[, 2:4], method = "euclidean")

as.matrix(euclidian_distance_matrix)
```

```
##          1         2         3         4
## 1 0.000000 1.4004970 1.1525624 1.1977190
## 2 1.400497 0.0000000 0.8465424 0.6453501
## 3 1.152562 0.8465424 0.0000000 1.0434277
## 4 1.197719 0.6453501 1.0434277 0.0000000
```

Answer: Based on the Euclidian distance metric using min-max normalization on the given data, customer C is most similar to customer A.

Part III. Data Manipulation in R

Question 7:

1. Import the "hills.csv" dataset.

```
hills = read.csv("hills.csv")
```

2. Use the dplyr function to arrange data based on "Time" and write down the race with the longest time.

```
longest_time = hills %>% arrange(Time)

longest_time
```

```
##                Race Distance Climb   Time
## 1       KildconHill      3.0   300 15.950
## 2       Greenmantle      2.5   650 16.083
## 3         BlackHill      4.5  1000 17.417
## 4           CowHill      2.0   900 17.933
## 5       NBerwickLaw      3.0   600 18.683
## 6            Acmony      5.0   500 20.950
## 7         CreagDubh      4.0  2000 26.217
## 8         EildonTwo      4.5  1500 26.933
## 9   MeallAnt-Suidhe      3.5  1500 27.900
## 10        Cockleroi      4.5   850 28.100
## 11         LargoLaw      5.0   950 28.567
## 12           Scolty      5.0   800 29.750
## 13      Knockfarrel      6.0   600 32.383
## 14        CreagBeag      5.5   600 32.567
## 15      CraigDunain      6.0   900 33.650
## 16        Burnswark      6.0   800 34.433
## 17      Cairnpapple      6.0   800 36.367
## 18         Traprain      6.0   650 39.750
## 19           Dollar      5.0  2000 43.050
## 20       CairnTable      6.0   500 44.133
## 21           BenRha      7.5   800 45.600
## 22     HalfBenNevis      6.0  2200 47.633
## 23         Carnethy      6.0  2500 48.350
## 24          Criffel      6.5  1750 50.500
## 25        BenLomond      8.0  3070 62.267
## 26          Lomonds      9.5  2200 65.000
## 27        Cairngorm     10.0  3000 72.250
## 28         Goatfell      8.0  2866 73.217
## 29        KnockHill      3.0   350 78.650
## 30         BenNevis     10.0  4400 85.583
## 31      SevenHills     14.0  2200  98.417
## 32     MoffatChase     20.0  5000 159.833
## 33    TwoBreweries     18.0  5200 170.250
## 34      LairigGhru     28.0  2100 192.667
## 35      BensofJura     16.0  7500 204.617
```

Answer: The race with the longest time is BensofJura.

3. Use dplyr function to get a subset of data with "Climb" less than 1000. Report the number of records in the subset.

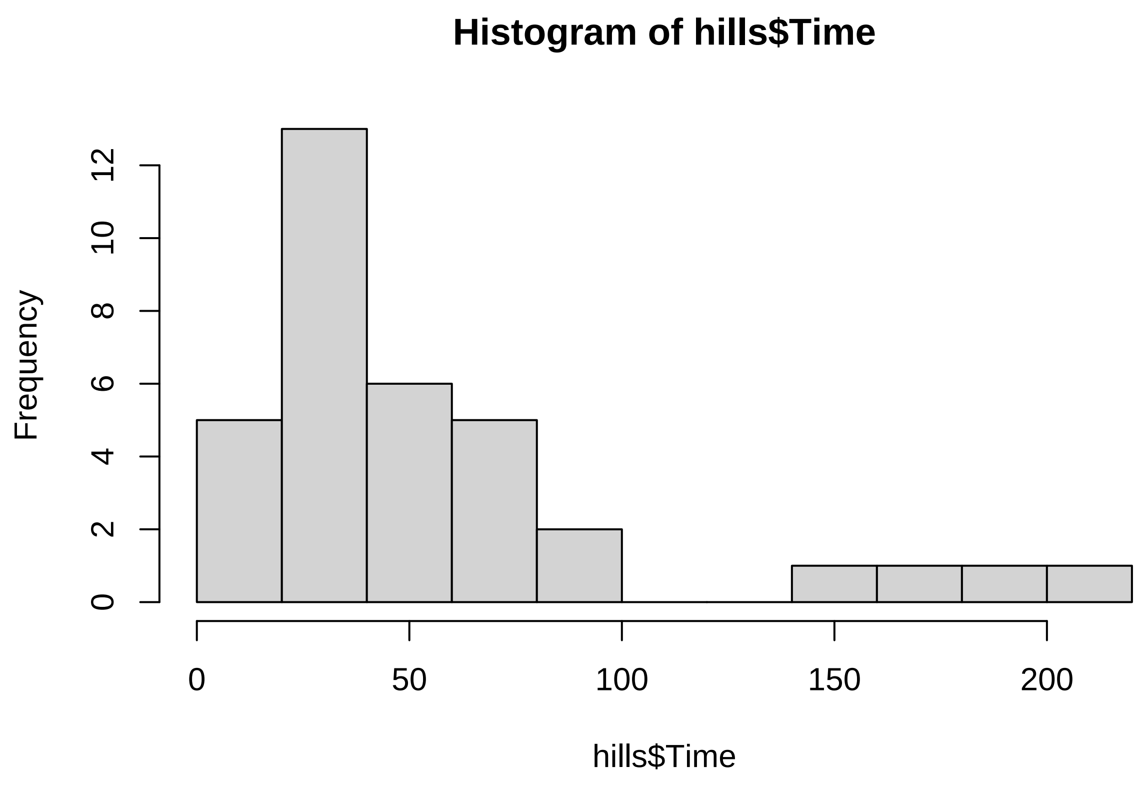
```r
data_climb = hills %>% filter(Climb < 1000)

nrow(data_climb)
```

```
## [1] 17
```

Answer: The number of records in the subset is 17.

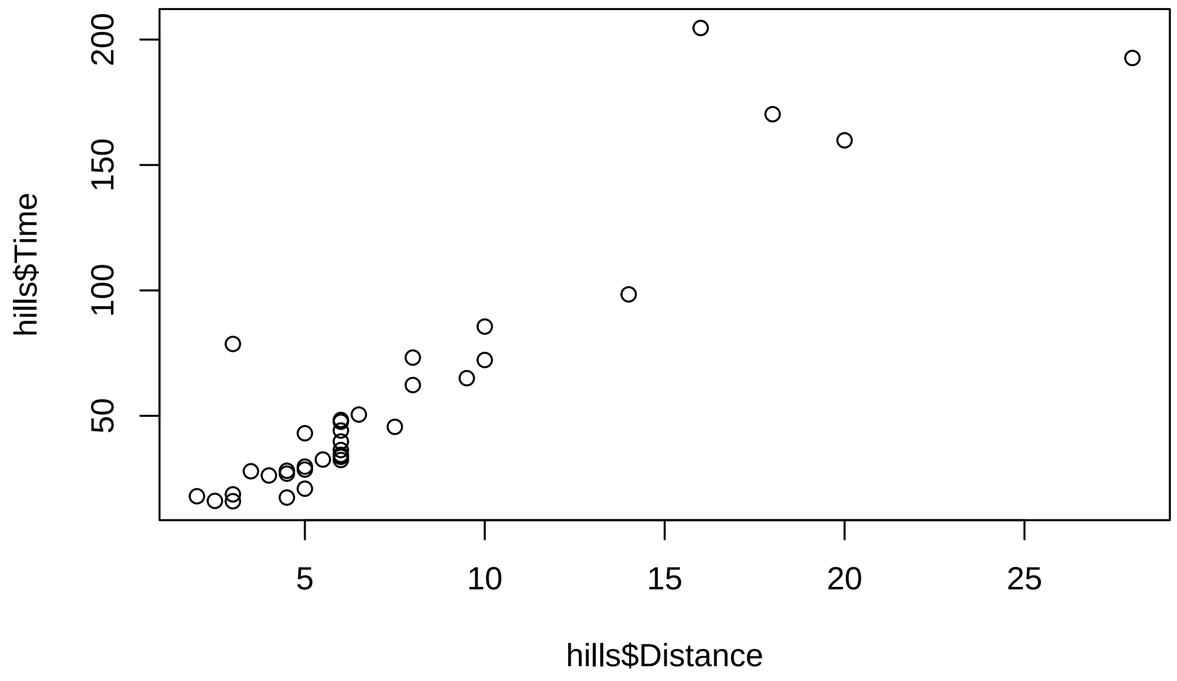
4. Create a histogram of the "Time" variable.

```r
hist(hills$Time)
```



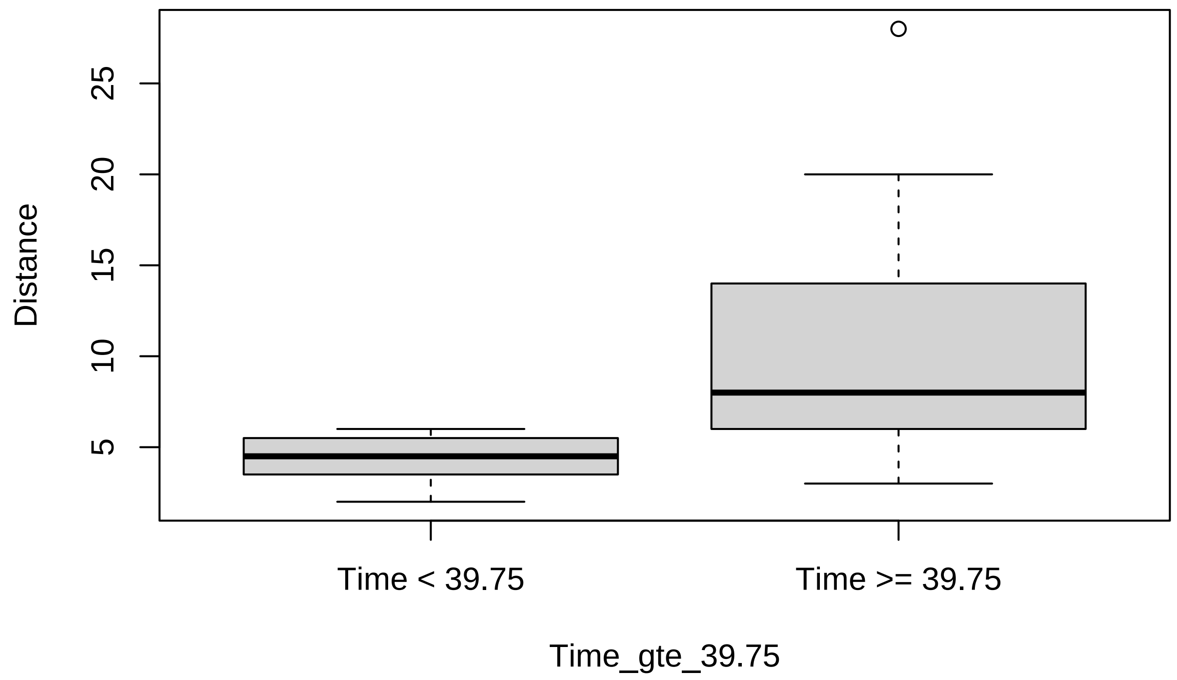
5. Create a scatterplot of the "Distance" and "Time" variables.

```r
plot(hills$Distance, hills$Time)
```



6. Create a boxplot of the "Distance" variable, grouped based on whether the "Time" for the race is $\geq$ 39.75 (or not). You should end up with two boxplots, side by side, in one chart.

```r
hills = hills %>% mutate(Time_gte_39.75 = ifelse(hills$Time >= 39.75, "Time >= 39.75", "Time < 39.75"))

boxplot(Distance ~ Time_gte_39.75, data = hills)
```



Then, interpret the components of either one of the two boxes.

In particular, interpret the following components:

(a) What does the thick black horizontal line represent?

Answer: The thick black horizontal line represents the median.

(b) What does the vertical height of the rectangle box in the chart represent?

Answer: The vertical height of the rectangle box in the chart represents the interquartile range (IQR) between the first (Q1) and third (Q3) quartiles where the middle fifty percent of the data is located.

(c) What do the two horizontal lines above and below the rectangle box (outside the box) represent?

Answer: The two horizontal lines above and below the rectangle box represent the maximum and minimum for the normal range of data (adjusted for outliers).

Part IV. Mine Association Rules from Data in R

Question 8:

1. Import the "groceries.csv" dataset as transaction data.

```
library(arules)

groceries = read.transactions("groceries.csv",
                              format = "basket",
                              sep = ",",
                              rm.duplicates = TRUE)
```

2. Find out the total number of unique items in this data.

```
nrow(itemInfo(groceries))
```

```
## [1] 169
```

Answer: There are 169 unique items in this data.

3. Find all association rules with a minsupp of 0.05 and a minconf of 0.05.

```
rules = apriori(groceries,
                parameter = list(supp = 0.05, conf = 0.05))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.05    0.1    1 none FALSE            TRUE       5    0.05      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: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [34 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

4. Inspect the rules and report the total number of rules found.

```
inspect(rules)
```

```
##      lhs                     rhs                        support    confidence
## [1]  {}                   => {canned beer}              0.07768175 0.07768175
## [2]  {}                  => {coffee}                0.05805796 0.05805796
## [3]  {}                  => {beef}                  0.05246568 0.05246568
## [4]  {}                  => {curd}                  0.05327911 0.05327911
## [5]  {}                  => {napkins}               0.05236401 0.05236401
## [6]  {}                  => {pork}                  0.05765125 0.05765125
## [7]  {}                  => {frankfurter}           0.05897306 0.05897306
## [8]  {}                  => {bottled beer}          0.08052872 0.08052872
## [9]  {}                  => {brown bread}           0.06487036 0.06487036
## [10] {}                  => {margarine}             0.05856634 0.05856634
## [11] {}                  => {butter}                0.05541434 0.05541434
## [12] {}                  => {newspapers}            0.07981698 0.07981698
## [13] {}                  => {domestic eggs}         0.06344687 0.06344687
## [14] {}                  => {fruit/vegetable juice} 0.07229283 0.07229283
## [15] {}                  => {whipped/sour cream}    0.07168277 0.07168277
## [16] {}                  => {pip fruit}             0.07564820 0.07564820
## [17] {}                  => {pastry}                0.08896797 0.08896797
## [18] {}                  => {citrus fruit}          0.08276563 0.08276563
## [19] {}                  => {shopping bags}         0.09852567 0.09852567
## [20] {}                  => {sausage}               0.09395018 0.09395018
## [21] {}                  => {bottled water}         0.11052364 0.11052364
## [22] {}                  => {tropical fruit}        0.10493137 0.10493137
## [23] {}                  => {root vegetables}       0.10899847 0.10899847
## [24] {}                  => {soda}                  0.17437722 0.17437722
## [25] {}                  => {yogurt}                0.13950178 0.13950178
## [26] {}                  => {rolls/buns}            0.18393493 0.18393493
## [27] {}                  => {other vegetables}      0.19349263 0.19349263
## [28] {}                  => {whole milk}            0.25551601 0.25551601
## [29] {yogurt}            => {whole milk}            0.05602440 0.40160350
## [30] {whole milk}        => {yogurt}                0.05602440 0.21925985
## [31] {rolls/buns}        => {whole milk}            0.05663447 0.30790492
## [32] {whole milk}        => {rolls/buns}            0.05663447 0.22164743
## [33] {other vegetables} => {whole milk}            0.07483477 0.38675775
## [34] {whole milk}        => {other vegetables}      0.07483477 0.29287704
##      coverage  lift     count
## [1]  1.0000000 1.000000  764
## [2]  1.0000000 1.000000  571
## [3]  1.0000000 1.000000  516
## [4]  1.0000000 1.000000  524
## [5]  1.0000000 1.000000  515
## [6]  1.0000000 1.000000  567
## [7]  1.0000000 1.000000  580
## [8]  1.0000000 1.000000  792
## [9]  1.0000000 1.000000  638
## [10] 1.0000000 1.000000  576
## [11] 1.0000000 1.000000  545
## [12] 1.0000000 1.000000  785
## [13] 1.0000000 1.000000  624
## [14] 1.0000000 1.000000  711
## [15] 1.0000000 1.000000  705
## [16] 1.0000000 1.000000  744
## [17] 1.0000000 1.000000  875
## [18] 1.0000000 1.000000  814
## [19] 1.0000000 1.000000  969
## [20] 1.0000000 1.000000  924
## [21] 1.0000000 1.000000 1087
## [22] 1.0000000 1.000000 1032
## [23] 1.0000000 1.000000 1072
## [24] 1.0000000 1.000000 1715
## [25] 1.0000000 1.000000 1372
## [26] 1.0000000 1.000000 1809
## [27] 1.0000000 1.000000 1903
## [28] 1.0000000 1.000000 2513
## [29] 0.1395018 1.571735  551
## [30] 0.2555160 1.571735  551
## [31] 0.1839349 1.205032  557
## [32] 0.2555160 1.205032  557
## [33] 0.1934926 1.513634  736
## [34] 0.2555160 1.513634  736
```

Answer: 34 association rules with a minsupp of 0.05 and a minconf of 0.05 were found.

5. Get a subset of rules whose LHS contain "whole milk". Report the subset of rules.

```r
inspect(subset(rules, lhs %pin% "whole milk"))
```

```
##     lhs               rhs                 support    confidence coverage lift
## [1] {whole milk} => {yogurt}            0.05602440 0.2192598  0.255516 1.571735
## [2] {whole milk} => {rolls/buns}        0.05663447 0.2216474  0.255516 1.205032
## [3] {whole milk} => {other vegetables} 0.07483477 0.2928770  0.255516 1.513634
##     count
## [1] 551
## [2] 557
## [3] 736
```

6. Report the rule with the highest lift (among the subset of rules in the last question). Interpret that rule. Also, interpret the lift ratio. Discuss what actions can be taken based on this rule.

Answer: The rule {whole milk} → {yogurt} has the highest lift, with a value of 1.571735.

Interpret this rule with the highest lift.

Answer: {whole milk} → {yogurt}, this rule with the highest lift, can be interpreted as customers who buy milk are likely to buy yogurt.

Interpret the lift ratio.

Answer: The lift of 1.571735 us that customers who buy whole milk are 0.571735x (57.1735%) more likely to buy yogurt than customers in general.

As a store manager, what action could you take after you know this rule?

Answer: As a store manager, knowing that customers who buy whole milk are likely to buy yogurt, I could situate whole milk closer to yogurt in my store, as a way to make shopping more convenient for my customers. Given the high likelihood that these two itemsets co-occur more than than pure chance, the location of these two items in my store should not matter. As a result, I could also place whole milk further away from yogurt in my store. I could even bundle these two items together as part of a breakfast promotion, given that these two items are typically consumed in the mornings. Finally, I could lower the price of whole milk and raise the price of yogurt, as I know there is a likelihood that whole milk buyers will still buy yogurt, despite the price increase.

Part V. Clustering Analysis in R

Question 9:

The "utility.csv" dataset contains information on 22 electricity power utilities. The first two columns are the name and ID of each utility, and the other columns (x1 - x8) are numeric characteristics of each utility.

1. Are all variables necessary for the purpose of clustering analysis? If so, explain why. If not, explain and select only the ones that are necessary.

Answer: No, not all variables are necessary for the purpose of clustering analysis. The name and ID variables are not necessary.

2. Is normalization necessary? Why or why not? If it is necessary, conduct normalization on selected variables.

Answer: Yes, there is a need to normalize the data. Column x6, compared to all other columns, varies greatly in range of values (3,300 - 17,441). If the data is not normalized, x6 would cause distortion in subsequent calculations.

```r
utility = read.csv("utility.csv")

normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
  }

utility_normalized = utility %>%
  mutate_at(c(3:10), normalize)
```

3. Get the distance matrix using **Manhattan** distance.

```r
distance_matrix = dist(utility_normalized[, 3:10], method = "manhattan")

as.matrix(distance_matrix)
```

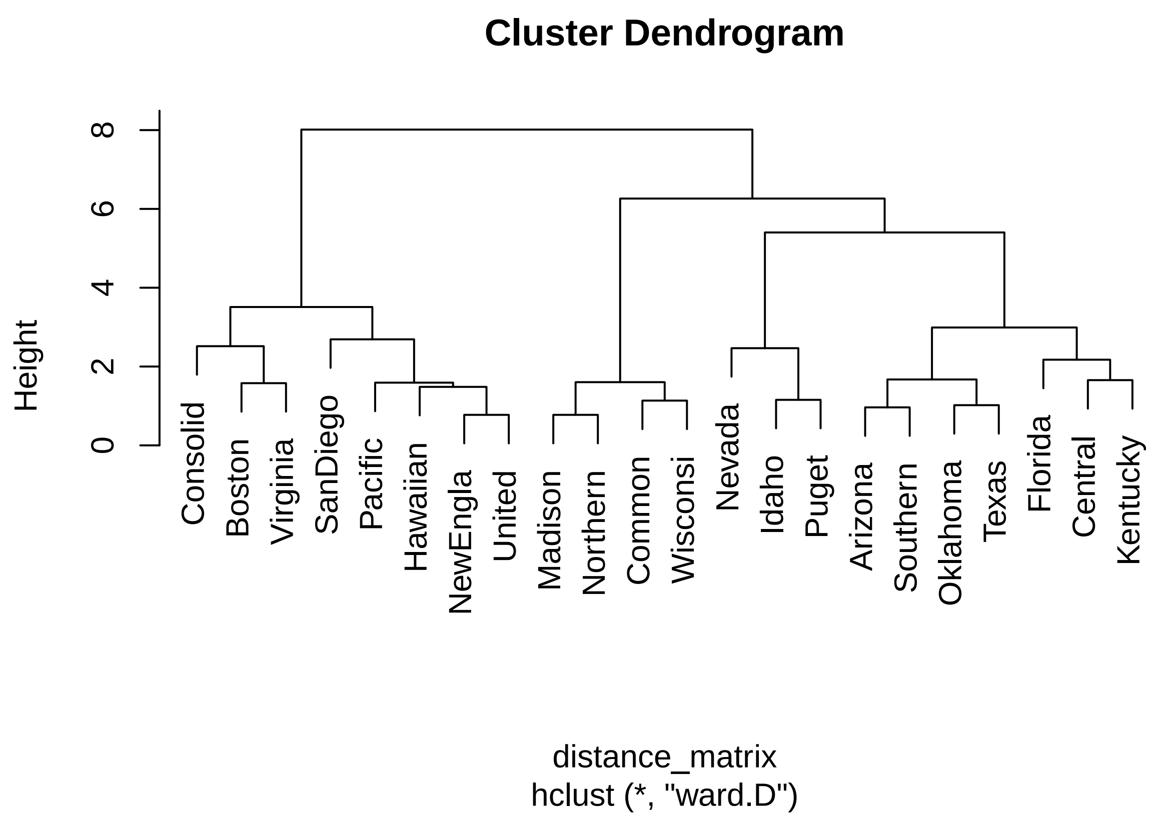
```
##           1        2        3         4        5        6         7        8
## 1  0.0000000 2.227212 1.916535 1.4999666 2.623658 2.665891 2.1657634 1.411563
## 2  2.2272124 0.000000 3.318113 1.5138818 2.113530 2.680514 2.1133516 2.587531
## 3  1.9165352 3.318113 0.000000 2.8923673 2.906962 1.941969 2.4020094 2.902651
## 4  1.4999666 1.513882 2.892367 0.0000000 2.723846 2.337464 2.5409155 2.513806
## 5  2.6236575 2.113530 2.906962 2.7238457 0.000000 3.181597 2.7129700 3.401471
## 6  2.6658915 2.680514 1.941969 2.3374637 3.181597 0.000000 2.4248697 3.387275
## 7  2.1657634 2.113352 2.402009 2.5409155 2.712970 2.424870 0.0000000 2.764128
## 8  1.4115631 2.587531 2.902651 2.5138056 3.401471 3.387275 2.7641281 0.000000
## 9  1.9148288 2.827232 1.655381 2.3980354 3.209887 2.145367 1.6302297 2.409955
## 10 1.8757131 1.784670 2.568689 0.9759161 2.661914 2.771917 2.5967363 2.390908
## 11 2.0105274 3.236013 3.279923 3.1047471 3.782449 4.231529 3.6340234 2.275356
## 12 1.9767101 1.675385 2.534562 2.2461224 2.247079 2.767948 0.9722923 2.343631
## 13 2.4686947 2.454407 2.969768 1.6712949 3.354863 3.217151 3.1086368 2.746732
## 14 1.0444697 3.027238 1.594125 2.0649041 3.192408 2.374856 2.5069298 2.246385
## 15 1.6132260 1.750934 3.053834 1.9460277 2.616755 2.637329 1.6785074 2.565988
## 16 2.1102632 3.243035 3.266090 3.0569502 3.938009 3.880617 3.2574701 1.155277
## 17 3.0012965 2.438202 3.933808 3.4276498 3.299755 3.836517 2.8827739 3.822479
## 18 0.9642114 2.024995 1.424368 1.7332411 3.056602 1.921690 1.5683507 1.774022
## 19 1.3789290 3.352842 1.909567 2.1975264 3.573255 1.758260 2.7100121 2.115946
## 20 1.9592235 2.100434 2.483448 1.1355818 3.247920 2.034202 2.1607304 2.672298
## 21 1.9659834 1.617507 3.090802 2.5912246 2.064615 3.324189 1.6408921 2.107679
## 22 1.5318073 1.578279 2.577047 1.6521519 2.450401 2.429867 2.4977715 2.179530
##           9        10        11        12        13       14       15       16
## 1  1.914829 1.8757131 2.010527 1.9767101 2.4686947 1.044470 1.613226 2.110263
## 2  2.827232 1.7846697 3.236013 1.6753848 2.4544075 3.027238 1.750934 3.243035
## 3  1.655381 2.5686893 3.279923 2.5345619 2.9697677 1.594125 3.053834 3.266090
## 4  2.398035 0.9759161 3.104747 2.2461224 1.6712949 2.064904 1.946028 3.056950
## 5  3.209887 2.6619136 3.782449 2.2470788 3.3548630 3.192408 2.616755 3.938009
## 6  2.145367 2.7719170 4.231529 2.7679485 3.2171508 2.374856 2.637329 3.880617
## 7  1.630230 2.5967363 3.634023 0.9722923 3.1086368 2.506930 1.678507 3.257470
## 8  2.409955 2.3909080 2.275356 2.3436308 2.7467319 2.246385 2.565988 1.155277
## 9  0.000000 2.4114361 2.692805 1.7101507 2.2075471 2.255995 2.333337 2.219086
## 10 2.411436 0.0000000 3.253903 2.4025104 0.7740305 2.054042 2.917418 2.879999
## 11 2.692805 3.2539035 0.000000 3.1818122 3.2764139 2.644861 2.676308 2.001224
## 12 1.710151 2.4025104 3.181812 0.0000000 2.8954236 2.562321 1.345884 2.847878
## 13 2.207547 0.7740305 3.276414 2.8954236 0.0000000 2.686937 3.610958 2.713815
## 14 2.255995 2.0540421 2.644861 2.5623209 2.6869371 0.000000 2.622608 2.824948
## 15 2.333337 2.9174180 2.676308 1.3458840 3.6109582 2.622608 0.000000 3.377047
## 16 2.219086 2.8799986 2.001224 2.8478784 2.7138148 2.824948 3.377047 0.000000
## 17 2.905618 3.9825133 2.808633 1.9489431 4.0287693 3.853458 1.886691 3.633539
## 18 1.413082 1.9358310 2.449433 1.5804670 2.4843681 1.289344 1.760716 2.395818
## 19 2.459077 2.4919806 2.709111 2.7798176 3.0038810 1.019872 2.685055 2.436819
## 20 1.789780 1.0870967 3.554514 2.0825875 1.3829491 2.306373 2.583547 3.130552
## 21 2.301479 2.7157133 3.018786 0.7738381 3.2175208 3.010453 1.285870 2.883651
## 22 2.166059 1.7994200 2.196013 2.0455602 1.8309981 2.431498 2.192095 2.399868
##          17        18       19       20        21       22
## 1  3.001296 0.9642114 1.378929 1.959224 1.9659834 1.531807
## 2  2.438202 2.0249949 3.352842 2.100434 1.6175068 1.578279
## 3  3.933808 1.4243677 1.909567 2.483448 3.0908022 2.577047
## 4  3.427650 1.7332411 2.197526 1.135582 2.5912246 1.652152
## 5  3.299755 3.0566016 3.573255 3.247920 2.0646153 2.450401
## 6  3.836517 1.9216896 1.758260 2.034202 3.3241887 2.429867
## 7  2.882774 1.5683507 2.710012 2.160730 1.6408921 2.497771
## 8  3.822479 1.7740221 2.115946 2.672298 2.1076788 2.179530
## 9  2.905618 1.4130824 2.459077 1.789780 2.3014786 2.166059
## 10 3.982513 1.9358310 2.491981 1.087097 2.7157133 1.799420
## 11 2.808633 2.4494334 2.709111 3.554514 3.0187859 2.196013
## 12 1.948943 1.5804670 2.779818 2.082588 0.7738381 2.045560
## 13 4.028769 2.4843681 3.003881 1.382949 3.2175208 1.830998
## 14 3.853458 1.2893444 1.019872 2.306373 3.0104530 2.431498
## 15 1.886691 1.7607161 2.685055 2.583547 1.2858702 2.192095
## 16 3.633539 2.3958184 2.436819 3.130552 2.8836508 2.399868
## 17 0.000000 2.9429836 4.179063 3.305002 1.9317344 2.697625
## 18 2.942984 0.0000000 1.615567 1.799523 1.9204910 1.605851
## 19 4.179063 1.6155669 0.000000 2.465011 3.3360579 2.697701
## 20 3.305002 1.7995235 2.465011 0.000000 2.6388278 1.966784
## 21 1.931734 1.9204910 3.336058 2.638828 0.0000000 2.119170
## 22 2.697625 1.6058511 2.697701 1.966784 2.1191705 0.000000
```

4. Apply hierarchical clustering using Ward's method.

```
hierarchical = hclust(distance_matrix, method = "ward.D")
```

5. Plot the dendrogram and report the number of clusters that you see fit.

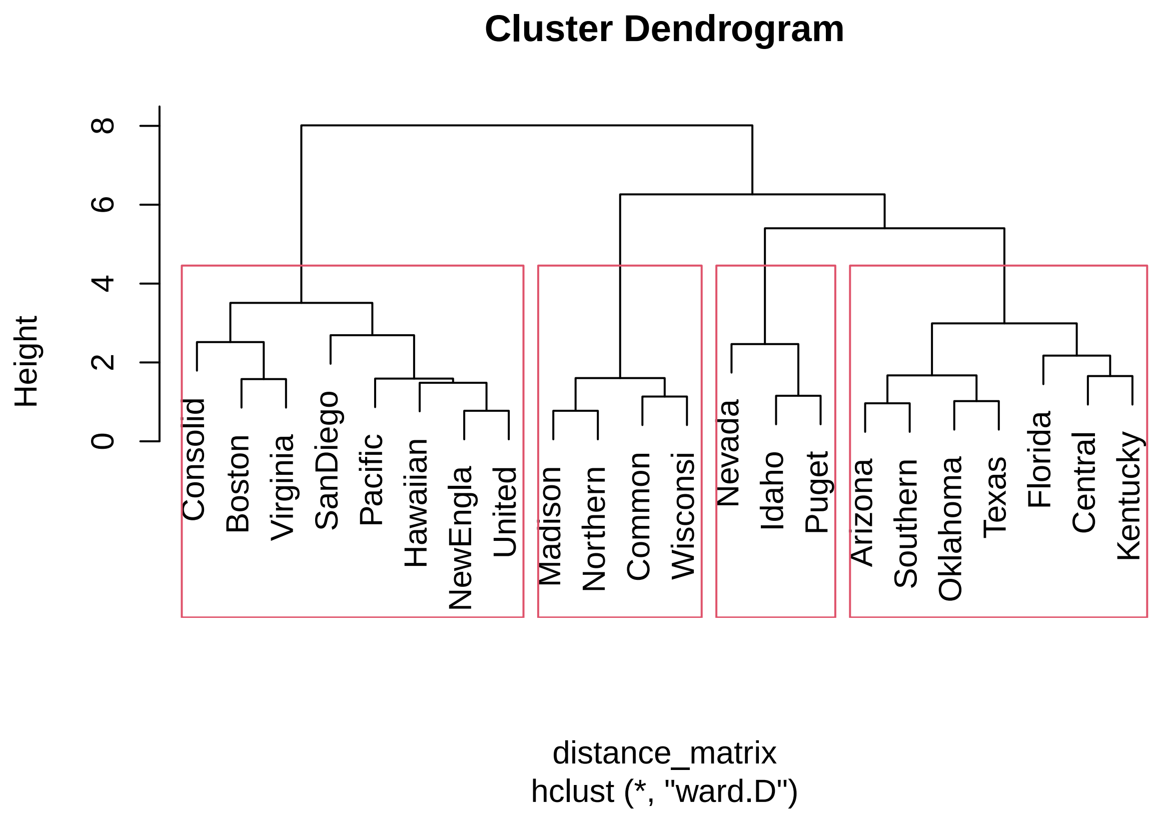
```
plot(hierarchical, labels = utility_normalized$utility_name)
```



Answer: I think that four clusters are appropriate.

6. Mark the clusters on the dendrogram, based on your answer to the last question.

```r
plot(hierarchical, labels = utility_normalized$utility_name)
rect.hclust(hierarchical, k = 4)
```



7. Apply K-Means clustering, using the number of clusters from the last two questions.

```r
kcluster = kmeans(utility_normalized[, 3:10], centers = 4)
```

8. Report the cluster centroids and interpret each cluster. Note that you do not have to differentiate the clusters on every single variable. Rather, try to describe each cluster by its most distinguishable characteristics. It is useful to know the meaning of each variable. Here they are:

- x1: Fixed - charge covering ration (income/debt)
- x2: Rate of return of capital
- x3: Cost per KW capacity in place
- x4: Annual Load Factor
- x5: Peak KWH demand growth from 1974 to 1975
- x6: Sales (KWH use per year)
- x7: Percent Nuclear
- x8: Total fuel costs (cents per KWH)

```r
kcluster$centers
```

```
##          x1        x2        x3        x4        x5        x6        x7
## 1 0.4324324 0.3174603 0.5302198 0.6035313 0.4624060 0.1809935 0.1425726
## 2 0.3423423 0.2740741 0.8162393 0.2827715 0.7485380 0.8630696 0.0000000
## 3 0.4891892 0.5644444 0.5205128 0.3134831 0.5228070 0.2961177 0.7625498
## 4 0.6177606 0.6761905 0.2023810 0.3186196 0.3433584 0.4853365 0.0640296
##          x8
## 1 0.8238596
## 2 0.1420402
## 3 0.2560044
## 4 0.3129101
```

Answer:

Cluster 1: Has high total fuel costs (cents per KWH) (x8) and annual load factor (x4).

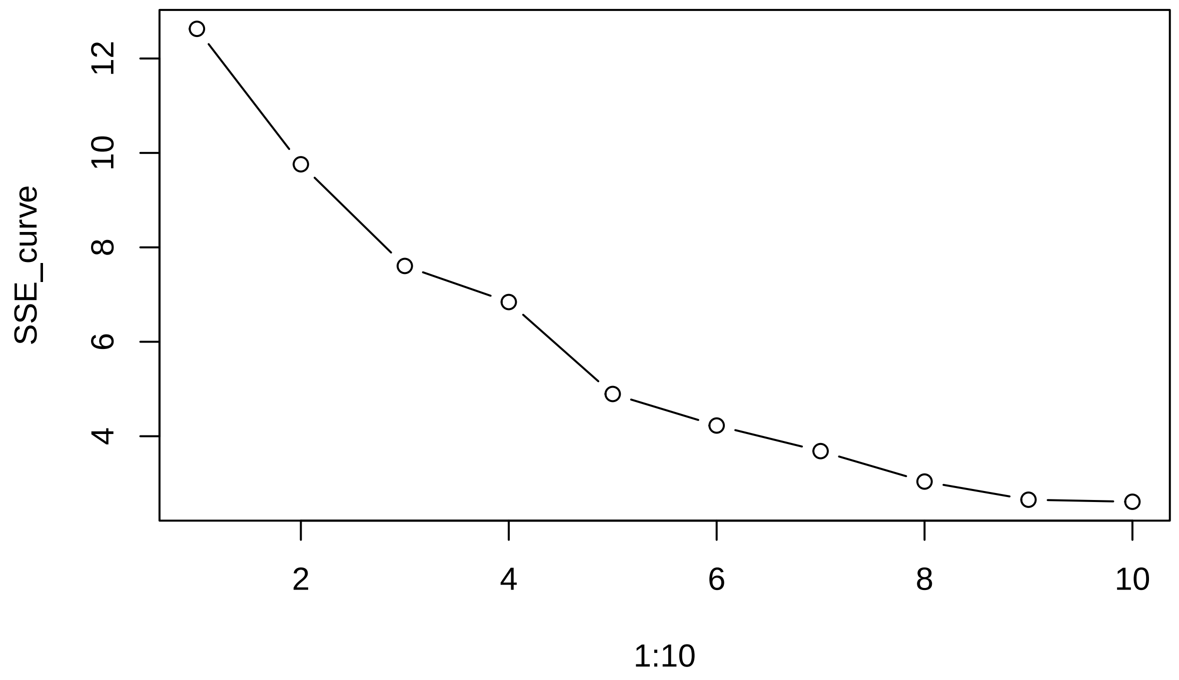
Cluster 2: Has high percent nuclear (x7), with every other variable being low-to-moderate.

Cluster 3: Has high cost per KW capacity in place (x3), peak KWH demand growth from 1974 to 1975 (x5), and sales (KWH use per year)(x6), with no percent nuclear (x7).

Cluster 4: High fixed - charge covering ration (income/debt) (x1) and rate of return of capital (x2).

9. Find the most natural number of clusters by plotting the SSE curve and explain *how* you found the cluster number.

```r
SSE_curve <- c()
for (n in 1:10) {
  kcluster = kmeans(utility_normalized[, 3:10], n)
  sse = kcluster$tot.withinss
  SSE_curve[n] = sse
  }

plot(1:10, SSE_curve, type = "b")
```



Answer: The most natural number of clusters in this data is four. You find the most natural number of clusters by looking for the "elbow" of the SSE plot. The elbow is the point where the SSE drops greatly before but very little after. Looking at the plot, the SSE drops greatly between one and four clusters and very little between four and ten clusters, which is why I decided four to be the most natural number of clusters in this data.