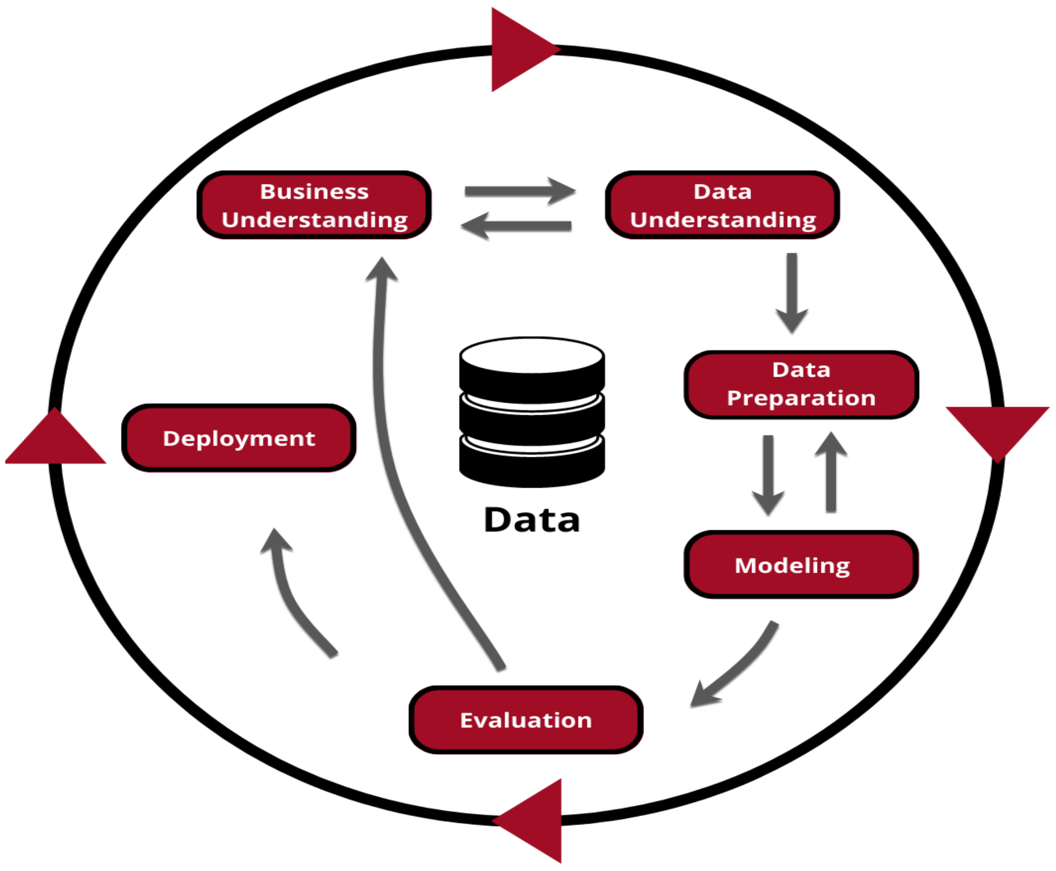
Follow Along: Download Your Data

Now that we have covered the basics of Association Rules and the steps, let's bring it together through a case study. For this Association Rules example, we will use transactional data. You can follow along using the Kaggle dataset below.

Market_Basket_Optimisation.csv

We will use the data mining process outlined previously.



We will follow the data mining process and Association Rules to analyze the relationship between items in transactional data. Let's start with the first step, which is understanding the business objectives and the purpose of data mining.

Business Understanding

Recall that this step clarifies the objectives and intended use of the data mining results. Determine if the analysis is a one-time event or an ongoing process. Understanding the purpose ensures alignment between the analytical outcomes and business goals. We must answer the following questions: How will the results of your work be used? Will this be a one-time event or ongoing? It is essential to set up a clear and concise problem statement.

> **Problem Statement: In this case study, we aim to understand the associations between items purchased together in transactions at a grocery store.**

Data Mining Objective

Why would a grocery store want to understand the associations?

Understanding the associations between items purchased together in transactions at a grocery store is essential for optimizing various aspects of retail operations. One significant advantage lies in optimizing product placement within the store. By discerning which items are frequently purchased together, grocery stores can strategically position these items near each other on shelves or in-store layouts. This strategic placement can capitalize on customer behavior, increasing sales through impulse purchases and enhanced customer convenience. For instance, placing pasta sauce next to pasta or chips next to salsa can prompt customers to purchase complementary items, boosting overall transaction value.

Identifying associations between items also enables grocery stores to capitalize on cross-selling opportunities. By recognizing which products tend to be purchased together, stores can develop targeted marketing strategies to promote complementary items. This can be achieved through bundling discounts or creating promotions that encourage customers to buy related products simultaneously. For instance, promoting bread alongside deli meats or offering discounts on cheese when purchasing wine can drive sales and enhance the shopping experience for customers.

In addition to optimizing sales and marketing efforts, understanding item associations aids in effective inventory management. Grocery stores can utilize market basket analysis to ensure that popular items are adequately stocked while minimizing waste and overstocking less popular items. By aligning inventory levels with customer demand and purchasing patterns, stores can optimize stocking levels, reduce holding costs, and improve overall profitability. This data-driven approach to inventory management enables stores to maintain optimal stock levels, ensuring that customers have access to the products they desire while minimizing excess inventory.

Ultimately, understanding associations between items purchased together enhances the overall shopping experience for customers. Grocery stores can improve customer satisfaction and loyalty by organizing products to make it easier for customers to find complementary items. A well-curated shopping experience, where customers can quickly locate and purchase related items, contributes to a positive shopping experience and encourages repeat visits. Therefore, leveraging market basket analysis to understand item associations is crucial for grocery stores seeking to optimize operations, increase sales, and enhance customer satisfaction.

In the next step, we will look at understanding the data.

Let's continue our case study with the next step. Now that you understand the business objectives and the purpose of the data mining process, the next phase involves obtaining relevant data from diverse sources. Remember to follow along using the dataset.

Market_Basket_Optimisation.csv

Data Understanding

In this step, we acquire relevant data from multiple sources or databases, ensuring it reflects the records of interest. The data in the CSV file contains transactional data. Transactional data in a CSV file is typically organized in a tabular format, where each row represents a transaction, and each column represents an item or attribute associated with that transaction.

Review the Dataset

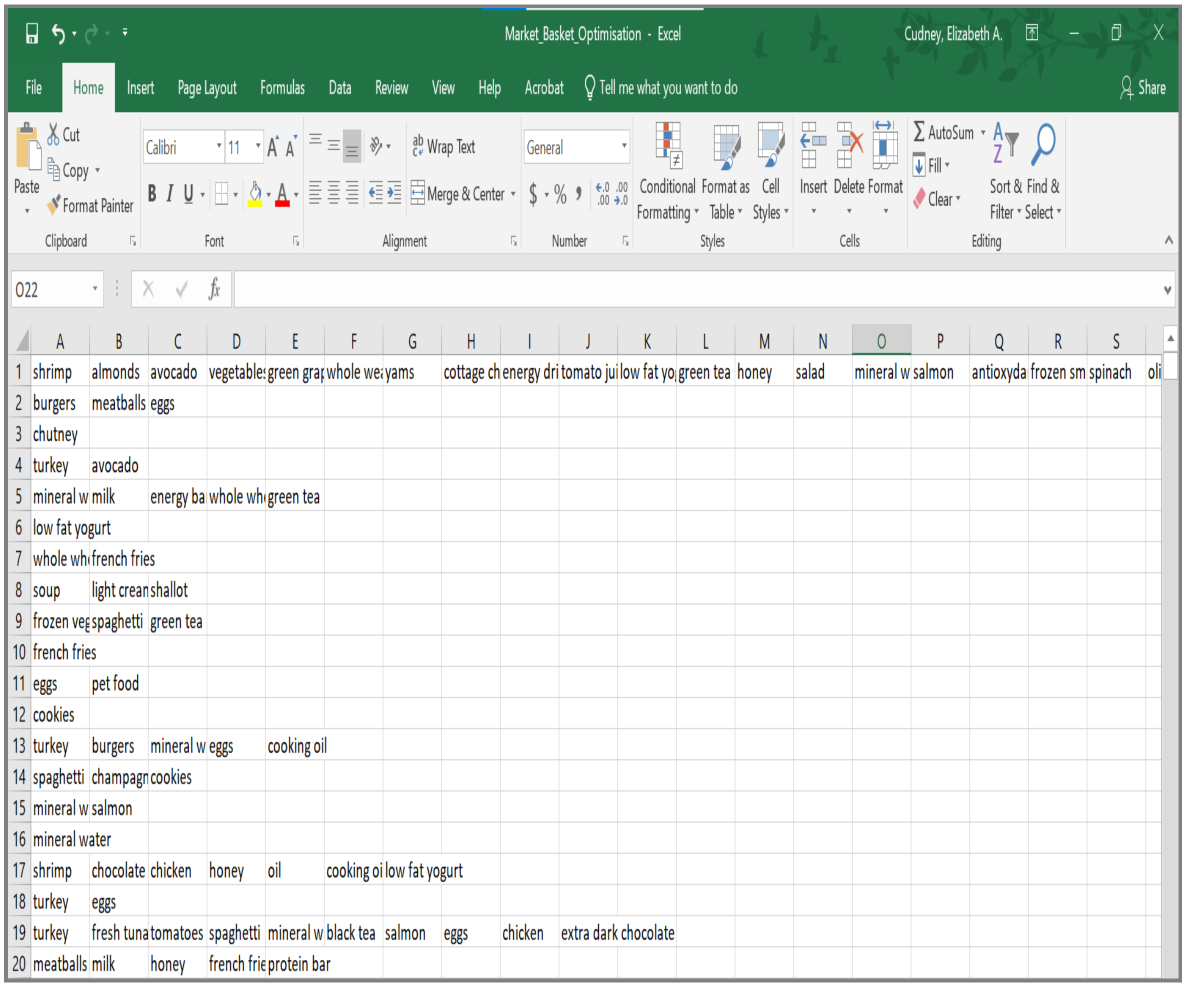
The dataset contains 7,501 transactions with 119 unique items. Each row represents information on a different transaction.

In transactional data used for market basket analysis, there typically is no traditional outcome variable in the same sense as supervised learning tasks. Instead, the focus is on understanding patterns and associations between items purchased together within transactions.

However, we can consider the presence or absence of specific items (or combinations of items) as the "outcome" or target variable of interest. For example, in association rule mining, we might be interested in discovering the rules of the form: "If item A is purchased, then item B is likely to be purchased as well." In this case, item B could be considered the outcome variable, and item A is the predictor or antecedent.

In some cases, additional information about transactions or customers may be included in the dataset, such as transaction timestamps, customer demographics, or purchase amounts. These variables can be used for segmentation, analysis, or prediction tasks, but they are not typically treated as outcome variables in market basket analysis. Instead, they provide additional context for understanding transaction patterns and customer behavior. Our case study contains only the items purchased in each transaction.

Our case study shows that the first transaction included quite a few grocery items. The second transaction had burgers, meatballs, and eggs. The third transaction was for only one item - chutney.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
1	shrimp	almonds	avocado	vegetables	green grap	whole wea	yams	cottage ch	energy dri	tomato jui	low fat yo	green tea	honey	salad	mineral w	salmon	antioxyda	frozen sm	spinach	oli
2	burgers	meatballs	eggs																	
3	chutney																			
4	turkey	avocado																		
5	mineral w	milk	energy ba	whole wh	green tea															
6	low fat yogurt																			
7	whole wh	french fries																		
8	soup	light crean	shallot																	
9	frozen veg	spaghetti	green tea																	
10	french fries																			
11	eggs	pet food																		
12	cookies																			
13	turkey	burgers	mineral w	eggs	cooking oil															
14	spaghetti	champagn	cookies																	
15	mineral w	salmon																		
16	mineral water																			
17	shrimp	chocolate	chicken	honey	oil	cooking oi	low fat yogurt													
18	turkey	eggs																		
19	turkey	fresh tuna	tomatoes	spaghetti	mineral w	black tea	salmon	eggs	chicken	extra dark chocolate										
20	meatballs	milk	honey	french frie	protein bar															

In transactional data used for market basket analysis, the variables represent the items that are being purchased. Therefore, we do not have a data dictionary for variables.

Each column in the dataset corresponds to an item or product, and each row represents a transaction. For example, in a grocery store dataset, variables could include items such as "milk," "bread," "eggs," "cheese," "fruit," "vegetables," and so on. In some datasets, each cell contains a value indicating whether the corresponding item was present (e.g., 1 for presence, 0 for absence) in the transaction.

Other than the items, there may not be traditional variables in the dataset, such as those typically found in structured datasets used for predictive modeling tasks. However, additional metadata or attributes about the transactions or customers could be included as different columns in the dataset. These could include transaction IDs, timestamps, customer IDs, transaction amounts, or other relevant information.

In transactional data for market basket analysis, the primary variables of interest are the items being purchased, while additional columns may contain supplementary information about the transactions or customers.

Now that we have obtained our data and understand our variables, we will move on to the next step, exploring and preparing the data.

Now that we have our data, we will continue our case study with the next step—data preparation. Remember to follow along using the dataset.

Market_Basket_Optimisation.csv

Data Preparation

In this step, we must organize our data to discern patterns easily. Data exploration involves examining the dataset's characteristics, such as size, structure, and completeness. Cleaning consists of handling missing values, removing duplicates, and addressing inconsistencies. Pre-processing includes cleaning the transactional data to remove inconsistencies, missing values, or irrelevant information. Ensure all transactions are correctly formatted, and the dataset is ready for analysis.

Since this is a new type of analysis with transactional data, we will start by loading the necessary libraries. We will load the `arules` and `arulesViz` libraries to use association rules for transactional data because these libraries provide functions and tools specifically designed for association rule mining and visualization.

Here's why each library is needed:

1. **arules**: This library contains functions for mining association rules from transactional data. It provides implementations of popular association rule mining algorithms such as Apriori and FP-growth, as well as functions for generating, manipulating, and evaluating association rules.

2. **arulesViz**: While the `arules` library provides functionality for mining association rules, the `arulesViz` library complements it by offering tools for visualizing and interpreting the results of association rule

mining. It provides functions for creating various plots and graphs to visualize association rules, itemsets, and other related data.

By loading both `arules` and `arulesViz` libraries, we can perform association rule mining on transactional data in R and then visualize the results to gain insights and inform decision-making in retail or other domains.

```
# Load necessary libraries
if (!requireNamespace("arules", quietly = TRUE)) {
  install.packages("arules")
}
# Load necessary libraries
if (!requireNamespace("arulesViz", quietly = TRUE)) {
  install.packages("arulesViz")
}
library(arules)
library(arulesViz)
```

Now that we have loaded the libraries let's go ahead and load the data into RStudio. Since we are using transactional data, the way we load the file differs from the processes we have used in other data mining techniques.

First, we will use the prompt `read.transactions` to read the transactional data into a special format for association rule mining. This function is from the `arules` package in R, commonly used for association rule mining. It reads transaction data from a CSV file and converts it into a transaction object.

Also, notice that we are reading the data and naming it a variable (MBO) instead of a data frame. In association rule mining, transaction data is typically represented in a special format called a transaction object rather than a regular data frame. This is because association rule mining algorithms and functions in R, such as those provided by the `arules` package, are designed to work with transaction objects.

Finally, because the data is comma delimited in the CSV file, we are specifying that the comma is being used as a separator in the CSV file. The parameter `sep = ','` specifies that a comma `,` is used as the separator in the CSV file. It tells R how to separate individual items in the transaction data.

Make sure to update the code below using the file path to where you saved the CSV file.

```
# Load the CSV data set and read transaction data into a special format
# for association rule mining into the variable MBO
# Specify that a comma is used as the separator in the CSV file
MBO <- read.transactions("C:\\Users\\ecudney\\OneDrive - Maryville University\\DATA 620 Data Mining\\Market_Basket_Optimisation.csv", sep = ',')
```

We will receive the warning message below when we run the code above. The warning message "removing duplicated items in transactions" indicates that the `read.transactions` function has detected and removed duplicated items within the transactions. This warning message refers explicitly to duplicate items within transactions, not duplicate transactions.

When working with transaction data for association rule mining, each transaction typically consists of a set of items. If the same item appears multiple times within a transaction, it may be considered a duplicate item.

How does this happen?

- **Quantity**: The same item might be purchased in multiple quantities in a single transaction. For example, if a customer buys two bottles of milk in one transaction, the item "milk" would appear twice.
- **Variants or Options**: Some items may have variants or options that are treated as separate items in the transaction dataset. For instance, if a

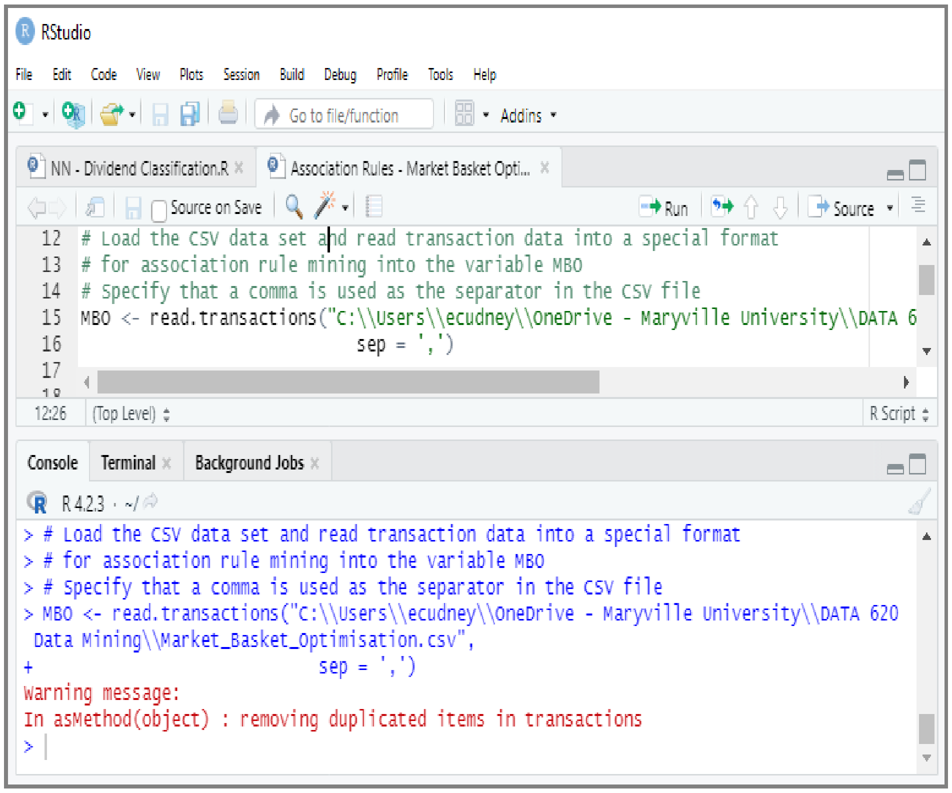
customer orders a pizza with extra cheese, the item "cheese" might appear multiple times in the transaction, once for the standard cheese and once for the extra cheese option.

- **Multi-pack Products**: Products sold in multi-pack or bundled form may result in multiple instances of the same item within a transaction. For example, if a customer buys a pack of six cans of soda, the item "soda" might appear six times in the transaction.

- **Repetitive Purchases**: Sometimes, customers may purchase the same item multiple times for different purposes or recipients. For example, if a customer buys two birthday cards for different people, the item "birthday card" would appear twice in the transaction.

In our case, the warning message suggests that there were duplicate items within some of the transactions in the dataset. The `read.transactions` function automatically removes these duplicate items to ensure consistency and accuracy in the transaction data.

While the warning message mentions removing duplicated items, it's important to note that this does not necessarily imply the removal of all duplicate transactions.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
NN - Dividend Classification.R
Association Rules - Market Basket Opti...
Source on Save
Run
Source
12:26
(Top Level)
R Script
Console
Terminal
Background Jobs
R 4.2.3 · ~/

```
# Load the CSV data set and read transaction data into a special format
# for association rule mining into the variable MBO
# Specify that a comma is used as the separator in the CSV file
MBO <- read.transactions("C:\\Users\\ecudney\\OneDrive - Maryville University\\DATA 6
                         sep = ',')
```

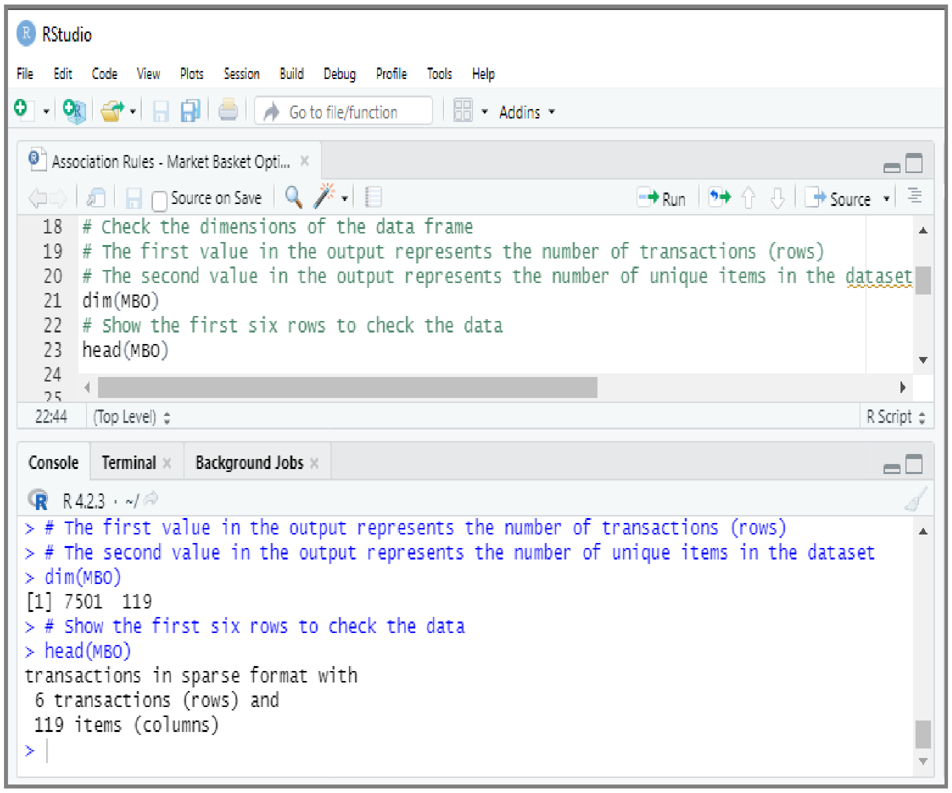
```
> # Load the CSV data set and read transaction data into a special format
> # for association rule mining into the variable MBO
> # Specify that a comma is used as the separator in the CSV file
> MBO <- read.transactions("C:\\Users\\ecudney\\OneDrive - Maryville University\\DATA 620
 Data Mining\\Market_Basket_Optimisation.csv",
+                          sep = ',')
Warning message:
In asMethod(object) : removing duplicated items in transactions
> 
```

Once we have loaded the dataset into RStudio, let's confirm all the data was loaded correctly by checking the data dimensions, viewing the first six rows, and showing all the data.

```
# Check the dimensions of the data frame
# The first value in the output represents the number of
transactions (rows)
# The second value in the output represents the number of unique
items in the dataset
dim(MBO)
# Show the first six rows to check the data
head(MBO)
```

Since the data contains 7,501 transactions, we expect the dimensions to match accordingly. This code calls the `dim` function on the transaction object `MBO`. The `dim` function returns the dimensions of an object, which for a transaction object includes the number of transactions (rows) and the number of unique items in the dataset. By printing the result of `dim(MBO)`, we will see two values: the number of transactions and the number of unique items. In our case, we have 7,501 transactions and 119 unique items purchased.

We also used the `head` function on the transaction object `MBO`. The `head(MBO)` command in R returns only the first few rows of the transaction object `MBO` because it is a sparse matrix representation of the transactional data. In sparse matrix format, the actual transaction data is stored efficiently using a sparse matrix structure, where only non-zero elements (i.e., transactions containing items) are explicitly stored. At the same time, the rest are assumed to be zero (i.e., transactions not containing items). The output indicates six (6) transactions (rows) and 119 items (columns) in the transactional data. Each row represents a transaction, and each column represents an item. The "sparse format" indicates that the transactional data is stored using a sparse matrix representation.



```
# Check the dimensions of the data frame
# The first value in the output represents the number of transactions (rows)
# The second value in the output represents the number of unique items in the dataset
dim(MBO)
# Show the first six rows to check the data
head(MBO)
```

```
> # The first value in the output represents the number of transactions (rows)
> # The second value in the output represents the number of unique items in the dataset
> dim(MBO)
[1] 7501  119
> # Show the first six rows to check the data
> head(MBO)
transactions in sparse format with
 6 transactions (rows) and
 119 items (columns)
>
```

Once you have confirmed that the data loaded correctly, let's explore the dataset further by obtaining the summarized data. We will use the `summary()` function in R to summarize the transactional data stored in the `MBO` object. When applied to transactional data, the `summary()` function typically includes information such as:

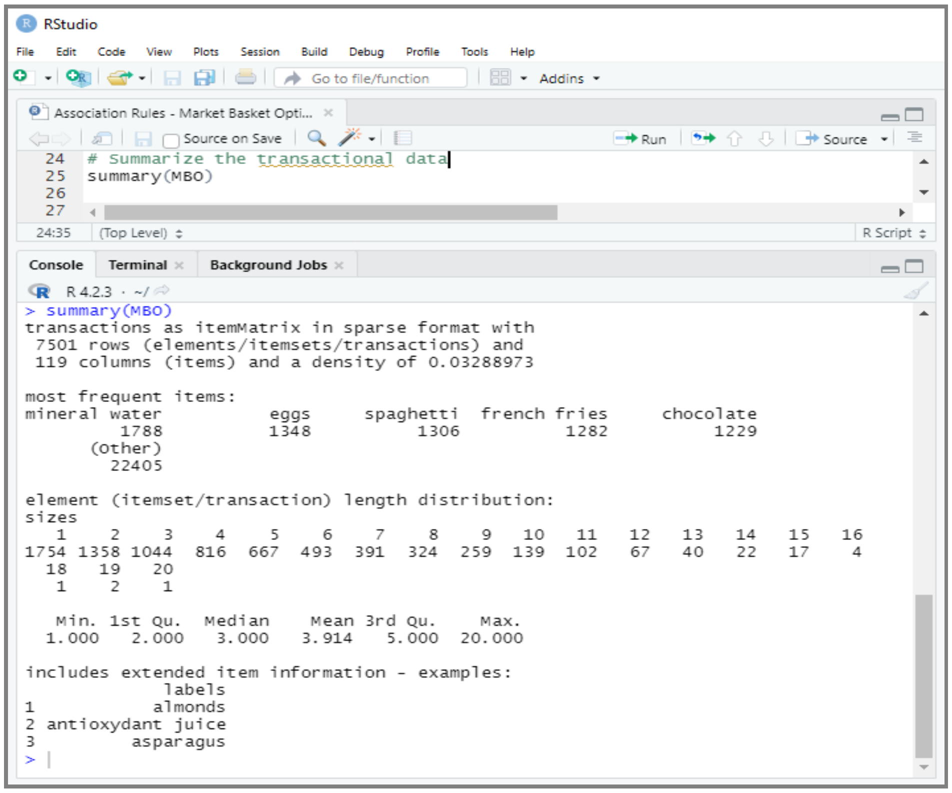
- The total number of transactions.
- The number of unique items.
- The most frequent items and their frequencies.
- Summary statistics for transaction sizes (number of items per transaction), including the minimum, 1st quartile, median, mean, 3rd quartile, and maximum transaction sizes.

```
# Summarize the transactional data
summary(MBO)
```

This provides the output below. In this example:

- There are 7,501 transactions and 119 unique items.
- The most frequent items include mineral water, eggs, spaghetti, french fries, and water. Their respective frequencies are provided.
- Summary statistics for transaction sizes are also provided, including the minimum, 1st quartile, median, mean, 3rd quartile, and maximum transaction sizes.

This summary provides a quick overview of the transactional data, including the distribution of item frequencies and transaction sizes, which can be helpful for exploratory data analysis and understanding the dataset's characteristics.



```
# Summarize the transactional data
summary(MBO)
```

```
> summary(MBO)
transactions as itemMatrix in sparse format with
 7501 rows (elements/itemsets/transactions) and
 119 columns (items) and a density of 0.03288973

most frequent items:
mineral water          eggs     spaghetti  french fries     chocolate
         1788          1348          1306          1282          1229
      (Other)
        22405

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17    4
  18   19   20
   1    2    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   3.914   5.000  20.000

includes extended item information - examples:
             labels
1           almonds
2 antioxydant juice
3         asparagus
> |
```

Next, let's determine the structure of the transactional data stored in the transaction object `MBO`. We will use the `str()` function in R to provide a compact display of the internal structure of our R object (`MBO`). It gives a concise summary of the type and structure of the object, including its class, length, and the first few elements, if applicable.

When we execute `str(MBO)`, it will display a summary of the structure of the transaction object `MBO`. This summary typically includes information such as:

- The class of the object (`transactions` in this case).
- The number of transactions (rows) and items (columns) in the transactional data.
- It may also provide a preview of the first few transactions and their corresponding items, if applicable.

We will use the following code:

```
# Determine structure using the str() function
str(MBO)
```

Using this code, we get the results below. The output of `str(MBO)` provides information about the structure of the transaction object `MBO`. Let's break down each part of the output:

- **Formal class 'transactions' [package "arules"] with 3 slots**: This indicates that `MBO` is an object of class 'transactions' from the 'arules' package. It has 3 slots, which are components of the object.

- **@ data**: This slot contains the transaction data in a sparse matrix format. It's represented as a formal class 'ngCMatrix' (a class from the 'Matrix' package). The sparse matrix has 5 slots:

    - `i`: A vector indicating the row indices of the non-zero elements in the sparse matrix.

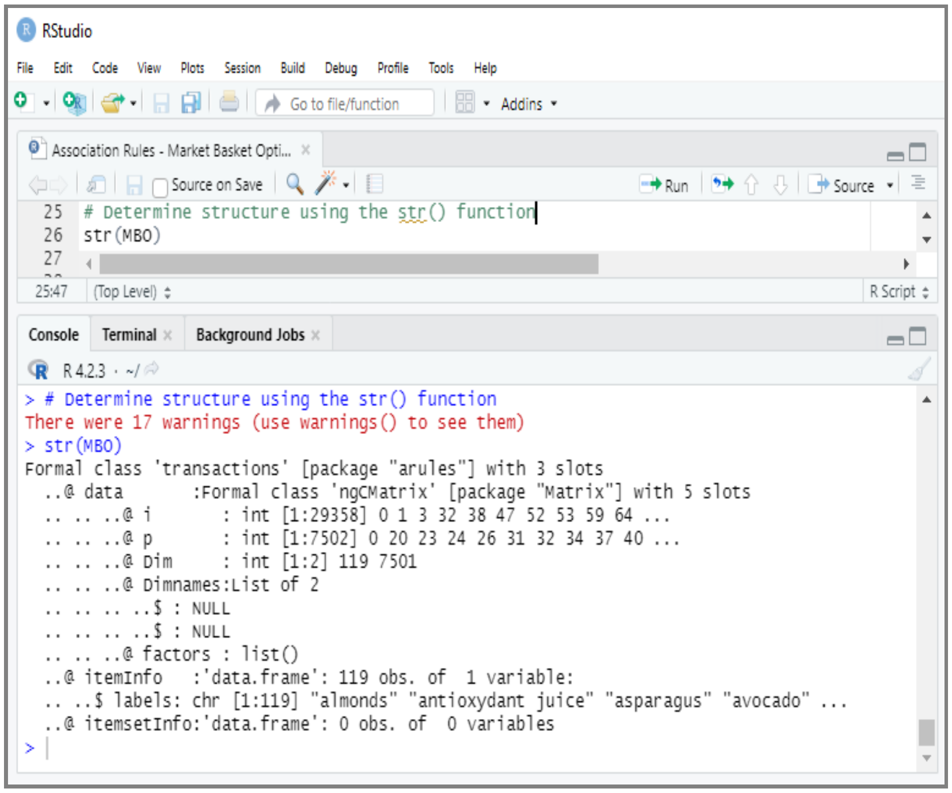
  - `p`: A vector of pointer values indicating the starting position of each column in the `i` vector.
  - `Dim`: A vector specifying the dimensions of the sparse matrix.
  - `Dimnames`: A list containing the row and column names of the sparse matrix (in this case, they are NULL, indicating that row and column names are not provided).
  - `factors`: A list of factors (categorical variables) used in the sparse matrix.

- **@ itemInfo**: This slot contains information about the items (items are the unique elements or products in the transactions). It's stored as a data frame with one variable:

  - `labels`: A character vector containing the labels (names) of the items. In this case, there are 119 unique items, such as "almonds," "antioxydant juice," "asparagus," etc.

- **@ itemsetInfo**: This slot contains information about itemsets, which are sets of items that occur together in transactions. However, in this case, 0 observations of 0 variables indicate no itemset information available.

Overall, this output provides a comprehensive summary of the structure of the transaction object `MBO`, including details about the transaction data, item information, and itemset information. It helps understand the content and format of the transactional data stored in the object.



```
# Determine structure using the str() function
str(MBO)
```

```
> # Determine structure using the str() function
There were 17 warnings (use warnings() to see them)
> str(MBO)
Formal class 'transactions' [package "arules"] with 3 slots
  ..@ data       :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
  .. .. ..@ i       : int [1:29358] 0 1 3 32 38 47 52 53 59 64 ...
  .. .. ..@ p       : int [1:7502] 0 20 23 24 26 31 32 34 37 40 ...
  .. .. ..@ Dim     : int [1:2] 119 7501
  .. .. ..@ Dimnames:List of 2
  .. .. .. ..$ : NULL
  .. .. .. ..$ : NULL
  .. .. ..@ factors : list()
  ..@ itemInfo   :'data.frame':  119 obs. of  1 variable:
  .. ..$ labels: chr [1:119] "almonds" "antioxydant juice" "asparagus" "avocado" ...
  ..@ itemsetInfo:'data.frame':  0 obs. of  0 variables
>
```

Next, we will look at the summary statistics for our items. Since we are working with transactional data, we will start by calculating item frequency for each item in the transaction object `MBO`. We will use the following code.

```
# Calculate item frequency
item_freq <- colSums(as(MBO, "matrix") > 0)
item_freq
```

The results below provide the frequency of each item purchased. The code calculates the item frequency by performing the following steps:

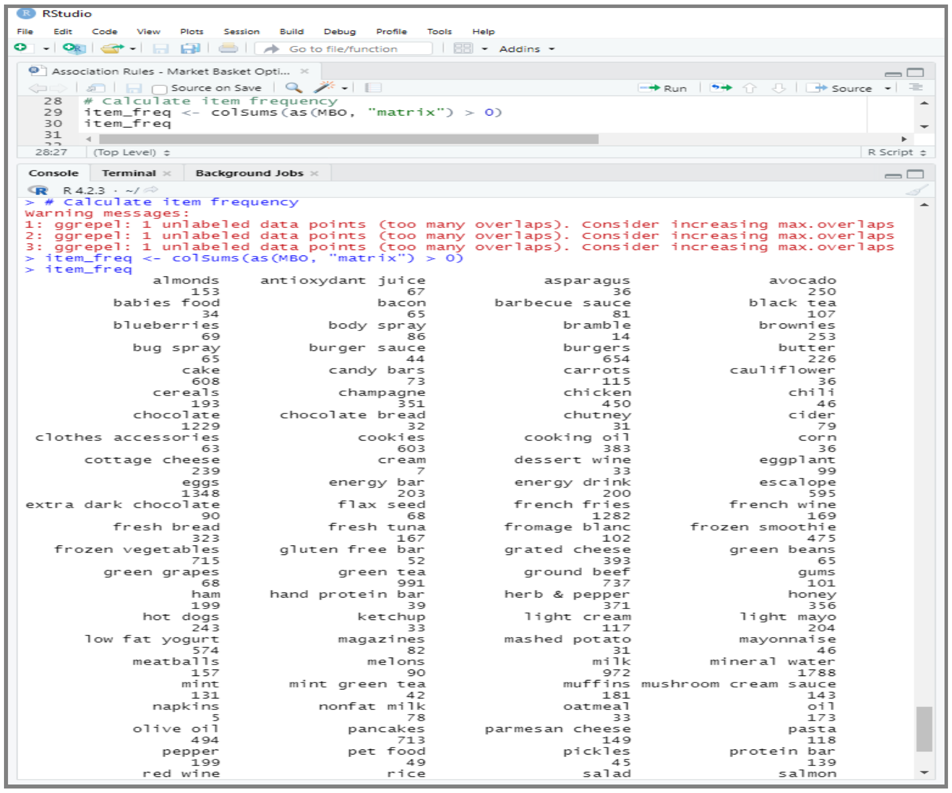
- `as(MBO, "matrix")`: Converts the transaction object `MBO` into a matrix format. In this matrix representation, each row represents a transaction, and each column represents an item. If an item is present in a transaction, the corresponding element in the matrix is set to `TRUE` (1), otherwise `FALSE` (0).

- `> 0`: Performs element-wise comparison to check if each element in the matrix is greater than 0. This effectively converts all non-zero values to `TRUE` and all zero values to `FALSE`.
- `colSums(...)`: Computes the column-wise sum of the resulting logical matrix. Since `TRUE` values are treated as 1 and `FALSE` values are treated as 0, the sum represents the frequency of each item across all transactions.

Let's interpret the data. For example, of the 7,501 transactions, 153 contained a purchase of almonds. Recall that we removed duplicates within a transaction. Therefore, we cannot state that 153 packages of almonds were purchased because a customer could have bought one package of lightly salted almonds and one package of wasabi almonds. This would have shown as a duplicate in our data. Therefore, we have to be careful with how we interpret the results. This is also why the term "frequency" is used instead of "count".

Finally, you may also be wondering about the warning message. The warning messages are from the `ggrepel` package, commonly used for creating text labels in plots to prevent overlap. These warnings suggest too many data points to label without overlap. However, we are simply calculating item frequency in this context and haven't yet created any plots or labels. Therefore, we can ignore these warning messages as they are not directly related to our calculation of item frequency.

The calculation of item frequency `item_freq <- colSums(as(MBO, "matrix") > 0)` should still have been executed successfully and `item_freq` should contain the frequency of each item in the dataset. We can proceed with using `item_freq`.



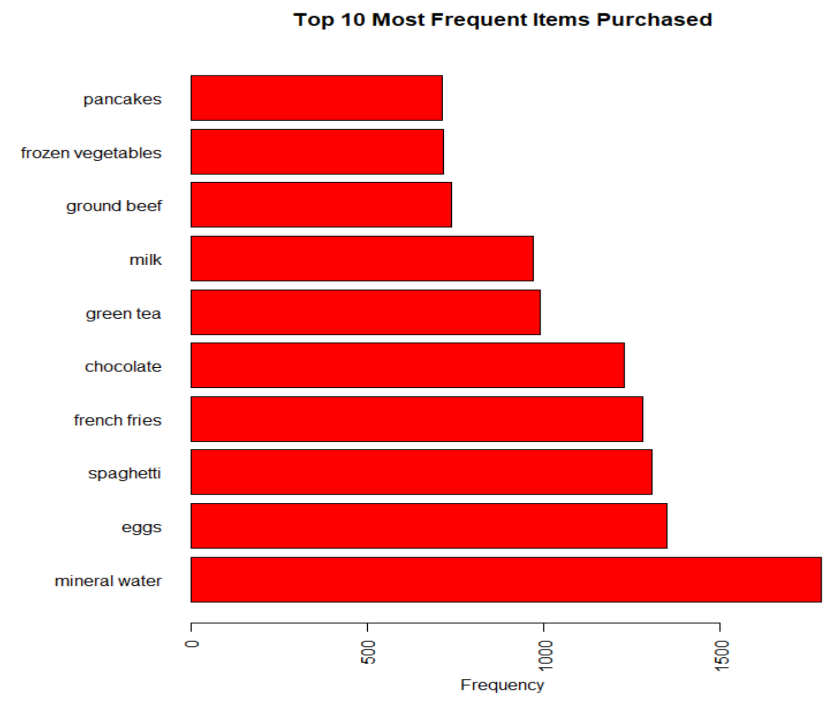
```r
# Calculate item frequency
item_freq <- colSums(as(MBO, "matrix") > 0)
item_freq
```

```
> # Calculate item frequency
Warning messages:
1: ggrepel: 1 unlabeled data points (too many overlaps). Consider increasing max.overlaps
2: ggrepel: 1 unlabeled data points (too many overlaps). Consider increasing max.overlaps
3: ggrepel: 1 unlabeled data points (too many overlaps). Consider increasing max.overlaps
> item_freq <- colSums(as(MBO, "matrix") > 0)
> item_freq
             almonds    antioxydant juice            asparagus              avocado
                 153                   67                   36                  250
         babies food                bacon       barbecue sauce            black tea
                  34                   65                   81                  107
         blueberries           body spray              bramble             brownies
                  69                   86                   14                  253
           bug spray         burger sauce              burgers               butter
                  65                   44                  654                  226
                cake           candy bars              carrots          cauliflower
                 608                   73                  115                   36
             cereals            champagne              chicken                chili
                 193                  351                  450                   46
           chocolate      chocolate bread              chutney                cider
                1229                   32                   31                   79
 clothes accessories              cookies          cooking oil                 corn
                  63                  603                  383                   36
      cottage cheese                cream         dessert wine             eggplant
                 239                    7                   33                   99
                eggs           energy bar         energy drink             escalope
                1348                  203                  200                  595
extra dark chocolate            flax seed         french fries          french wine
                  90                   68                 1282                  169
         fresh bread           fresh tuna        fromage blanc      frozen smoothie
                 323                  167                  102                  475
   frozen vegetables      gluten free bar        grated cheese          green beans
                 715                   52                  393                   65
        green grapes            green tea          ground beef                 gums
                  68                  991                  737                  101
                 ham     hand protein bar        herb & pepper                honey
                 199                   39                  371                  356
            hot dogs              ketchup          light cream           light mayo
                 243                   33                  117                  204
      low fat yogurt            magazines        mashed potato           mayonnaise
                 574                   82                   31                   46
           meatballs               melons                 milk        mineral water
                 157                   90                  972                 1788
                mint       mint green tea              muffins mushroom cream sauce
                 131                   42                  181                  143
             napkins          nonfat milk              oatmeal                  oil
                   5                   78                   33                  173
           olive oil             pancakes      parmesan cheese                pasta
                 494                  713                  149                  118
              pepper             pet food              pickles          protein bar
                 199                   49                   45                  139
            red wine                 rice                salad               salmon
```

Let's start by creating some visualization to understand our data better. Let's start by creating a bar chart of the 10 most frequently purchased items. We will use the following code:

> # Plot item frequency of top 10 most frequently purchases items
> itemFrequencyPlot(MBO, topN = 10, type = "absolute", col = "red", horiz = TRUE,
>   main = "Top 10 Most Frequent Items Purchased", xlab = "Frequency")

This code provides the bar chart below. The bar chart indicates that the top 10 most frequently purchased items are mineral water, eggs, spaghetti, french fries, chocolate, green tea, milk, ground beef, frozen vegetables, and pancakes.



We can create a similar plot by looking at relative frequency. The purpose of a relative frequency plot is to visualize a dataset's distribution in terms of proportions or percentages rather than absolute counts. Relative frequency plots are particularly useful when comparing groups or categories with different sample sizes.

By representing data as proportions or percentages, relative frequency plots allow for a more intuitive comparison of the relative importance or occurrence of different categories within the dataset. Relative frequency plots allow analysts to observe the proportion of transactions containing specific items or itemsets relative to the total number of transactions. This visualization can help identify frequent items or itemsets, which are crucial for association rule mining.

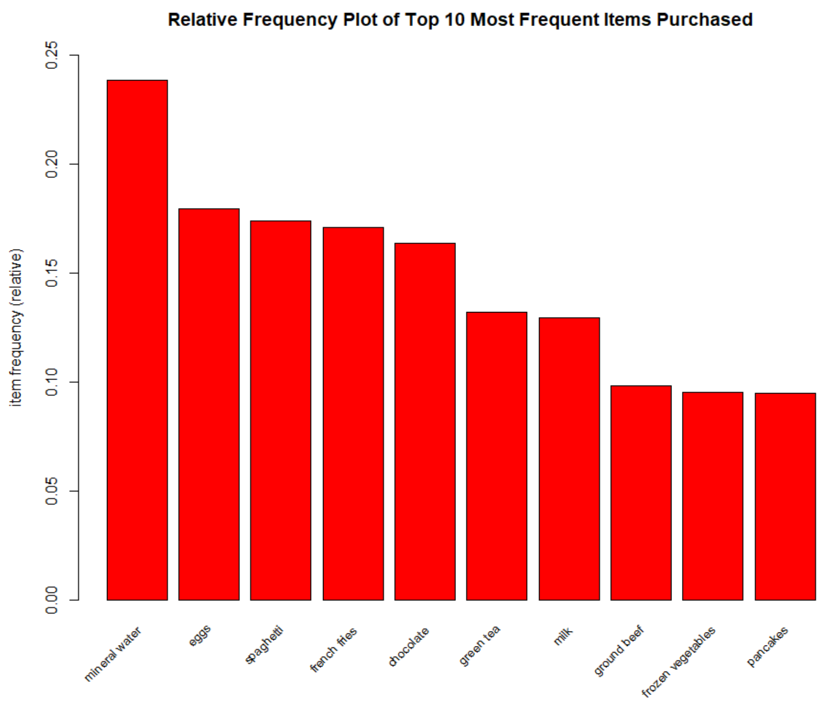
It is important to note that some transactional data is binary. If the data is binary, then the data represents the presence or absence of items in transactions. Relative frequency plots are not directly applicable to this type of transactional data. However, in our example, the data is not. Our data represents item occurrences in transactions with counts or frequencies.

Therefore, relative frequency plots help visualize the distribution of item occurrences across transactions.

We will use the following code:

```
# Relative frequency plot of top 10 most frequently purchases
items
itemFrequencyPlot(MBO, topN = 10, main = "Relative Frequency
Plot of Top 10 Most Frequent Items Purchased",
                cex.names = 0.8,  horiz = FALSE, col="red",
ylim=c(0, 0.25))
```

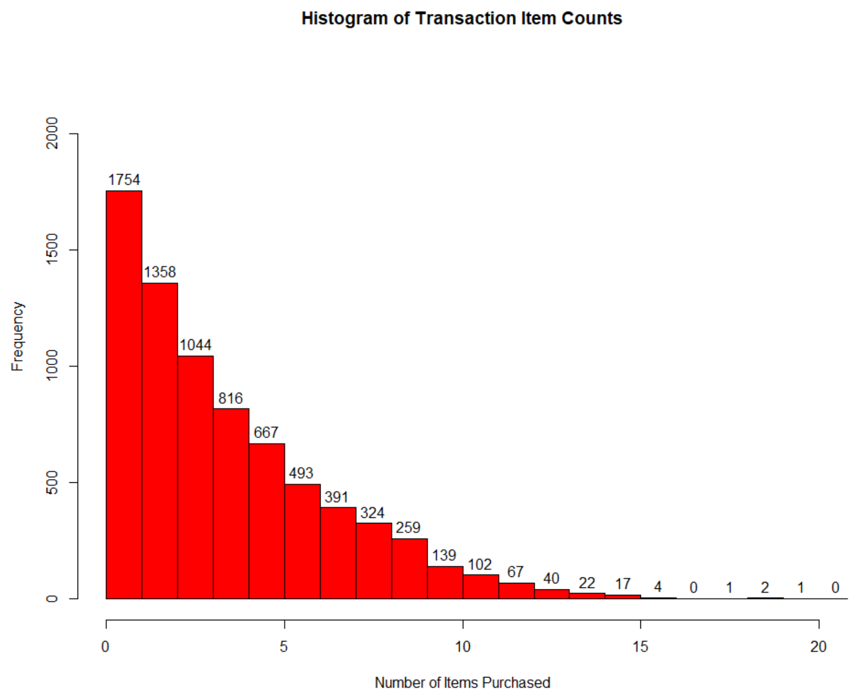
This code provides the bar chart below. The bar chart indicates that the top 10 most frequently purchased items are mineral water, eggs, spaghetti, french fries, chocolate, green tea, milk, ground beef, frozen vegetables, and pancakes. This is the same order as the previous bar chart; however, the data is now provided as a proportion rather than a count.



Next, let's construct a visualization to illustrate the distribution of transaction sizes (the number of items per transaction). The following code will generate a histogram to visualize the number of items in each purchase:

```
# Create a histogram based on transaction item counts
# Calculate the number of items purchased in each transaction
transaction_counts <- size(MBO)
transaction_counts
# Determine breaks for the histogram
breaks <- seq(0, max(transaction_counts) + 1, by = 1)
# Create a histogram of the transaction item counts
hist(transaction_counts,
    main = "Histogram of Transaction Item Counts",
    xlab = "Number of Items Purchased",
    ylab = "Frequency",
    col = "red",
    border = "black",
    breaks = breaks,  # Specify breaks to match those used in cut()
function
    xlim = c(0, max(transaction_counts)),  # Set the x-axis limits
from 0 to the maximum transaction count
    ylim = c(0, max(freq_counts) + 500),  # Set the y-axis limits to
one level higher than the maximum frequency count
    labels = TRUE  # Add counts to the bars
)
```

This provides the histogram below. The histogram shows the frequency distribution of the number of items purchased in each transaction. The histogram indicates that around 1,754 transactions included purchasing only one (1) item. In addition, 1,358 customers purchased two (2) items. The number of items purchased per transaction then continues to drop slowly.



We can look closer by creating a matrix of the frequency counts and percentages. We will calculate the frequency counts for each bin in the histogram created from the transaction item counts. Then, we will use the `cut` function to discretize the transaction counts into bins and then the `table` function to count the occurrences of each bin. Finally, we will combine the frequency counts and percentages into a matrix using the `cbind` function. This matrix contains two columns: one for the frequency counts and the other for the percentages.

```
# Create a matrix of the frequency counts
# Get the frequency counts for each bin in the histogram
freq_counts <- table(cut(transaction_counts, breaks = breaks))
# Calculate percentages
percentages <- prop.table(freq_counts) * 100
# Calculate cumulative percentages
cumulative_percentages <- cumsum(percentages)
# Create a matrix containing frequencies, percentages, and
cumulative percentages
frequency_matrix <- cbind(freq_counts, percentages,
cumulative_percentages)
# Print the matrix
print(frequency_matrix)
```

This provides the matrix output shown below. These results represent frequency counts and percentages for data within different ranges or bins. The data has been grouped into intervals, and the frequency count indicates how many data points fall within each interval. The percentages represent the proportion of the total data that each interval represents. The cumulative percentages indicate the cumulative proportion of the total dataset covered by each bin and all previous bins.

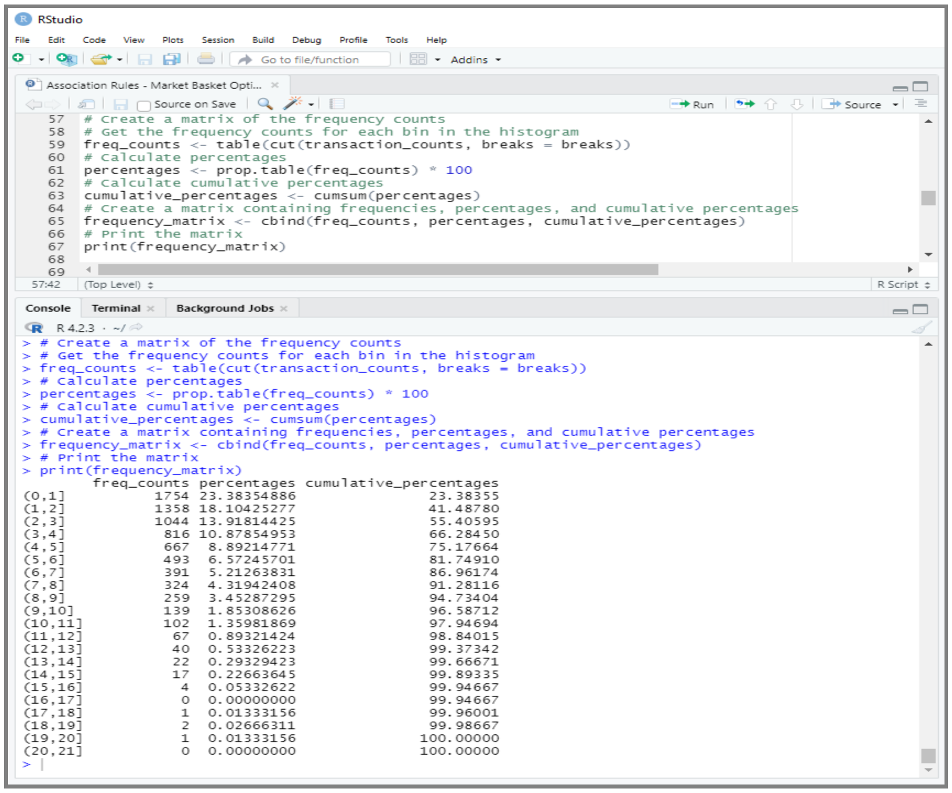
For example:

- The interval (0,1] contains 1,754 data points, approximately 23.38% of the total data. This means that nearly a quarter of the observations fall within this range.
- The interval (1,2] contains 1,358 data points, approximately 18.10% of the total data. This indicates that around 18% of the observations fall within this range.
- The interval (2,3] contains 1,044 data points, approximately 13.92% of the total data. This implies that nearly 14% of the observations are within this range.
- And so on...

Each interval represents a range of values, and the frequency count tells you how many data points fall within that range, while the percentage tells you the proportion of the total data that corresponds to that range.

For instance, the interval (0,1] indicates values greater than 0 and less than or equal to 1. Therefore, the frequency count of 1,754 means 1,754 data points fall within this range, representing 23.38% of the total data. Similarly, the interval (1,2] indicates values greater than 1 and less than or equal to 2, and so on for the other intervals.

The cumulative percentage results further illustrate the distribution of the data. After the bin (1,2], approximately 41.49% of the total observations have been accounted for. This means that the sum of data points falling within the bins up to (1,2] represents 41.49% of the entire dataset. Similarly, after the bin (2,3], approximately 55.41% of the observations have been captured, indicating that the cumulative proportion of the dataset up to this point is 55.41%. This cumulative trend continues, with each subsequent bin adding to the total percentage of observations covered until reaching 100% at the end.



```
# Create a matrix of the frequency counts
# Get the frequency counts for each bin in the histogram
freq_counts <- table(cut(transaction_counts, breaks = breaks))
# Calculate percentages
percentages <- prop.table(freq_counts) * 100
# Calculate cumulative percentages
cumulative_percentages <- cumsum(percentages)
# Create a matrix containing frequencies, percentages, and cumulative percentages
frequency_matrix <- cbind(freq_counts, percentages, cumulative_percentages)
# Print the matrix
print(frequency_matrix)
```

```
> # Create a matrix of the frequency counts
> # Get the frequency counts for each bin in the histogram
> freq_counts <- table(cut(transaction_counts, breaks = breaks))
> # Calculate percentages
> percentages <- prop.table(freq_counts) * 100
> # Calculate cumulative percentages
> cumulative_percentages <- cumsum(percentages)
> # Create a matrix containing frequencies, percentages, and cumulative percentages
> frequency_matrix <- cbind(freq_counts, percentages, cumulative_percentages)
> # Print the matrix
> print(frequency_matrix)
        freq_counts percentages cumulative_percentages
(0,1]          1754 23.38354886               23.38355
(1,2]          1358 18.10425277               41.48780
(2,3]          1044 13.91814425               55.40595
(3,4]           816 10.87854953               66.28450
(4,5]           667  8.89214771               75.17664
(5,6]           493  6.57245701               81.74910
(6,7]           391  5.21263831               86.96174
(7,8]           324  4.31942408               91.28116
(8,9]           259  3.45287295               94.73404
(9,10]          139  1.85308626               96.58712
(10,11]         102  1.35981869               97.94694
(11,12]          67  0.89321424               98.84015
(12,13]          40  0.53326223               99.37342
(13,14]          22  0.29329423               99.66671
(14,15]          17  0.22663645               99.89335
(15,16]           4  0.05332622               99.94667
(16,17]           0  0.00000000               99.94667
(17,18]           1  0.01333156               99.96001
(18,19]           2  0.02666311               99.98667
(19,20]           1  0.01333156              100.00000
(20,21]           0  0.00000000              100.00000
> 
```

Now that the data has been cleaned and processed and we have explored our data, we can use it for modeling. Next, we will determine the data mining task and select the appropriate algorithm.

Now that we have cleaned and explored the data, we will determine the data mining task and appropriate method and then apply the selected method. Continue to follow along using the dataset.

Market_Basket_Optimisation.csv

Modeling

This step defines the specific data mining task based on the business problem. Decide whether classification, prediction, clustering, or other techniques can extract the desired insights from the data. Will you mine the data using classification, prediction, clustering, etc.?

Determine the Data Mining Task

In this case, we aim to understand the associations between items purchased together in transactions at a grocery store. The data is transactional data. Therefore, our task is unsupervised learning or descriptive modeling because we will identify frequently co-occurring items in transactions.

Choose the Methods

Now that we have determined that our data mining task is unsupervised learning, we can choose our method for the analysis. Here, we select appropriate data mining algorithms or methods based on the chosen task.

Given our task is unsupervised learning and our data is transactional, we will select an appropriate method. We will use Association Rules for this example.

Clean and Transform the Data Set

Since we are using association rules, our first step is calculating item support, particularly in market basket analysis. This is to calculate item support because it helps identify frequently occurring itemsets in the dataset. Item support measures how frequently an itemset appears in the dataset relative to the total number of transactions.

Here's why calculating item support is an important initial step:

- **Identifying frequently occurring items**: High-support itemsets represent combinations of items that frequently co-occur together in transactions. These itemsets are candidates for forming strong association rules. By calculating item support, you can quickly identify these frequently occurring itemsets.
- **Filtering infrequent itemsets**: Items that appear infrequently may not provide meaningful insights for association rule mining. Setting a minimum support threshold allows you to filter out infrequent itemsets and focus on the most relevant patterns. This threshold helps in reducing the search space and computational complexity of the mining process.
- **Basis for rule generation**: Item support is the basis for generating association rules. Once high-support itemsets are identified, association rules can be derived from them by calculating other metrics, such as confidence and lift. These rules indicate the likelihood of one item being purchased, given the presence of another item.
- **Insight into customer behavior**: Understanding item support helps businesses gain insights into customer behavior and purchasing patterns. It allows them to identify popular products, discover item associations, and tailor marketing strategies accordingly.

We will calculate item support using the following code:

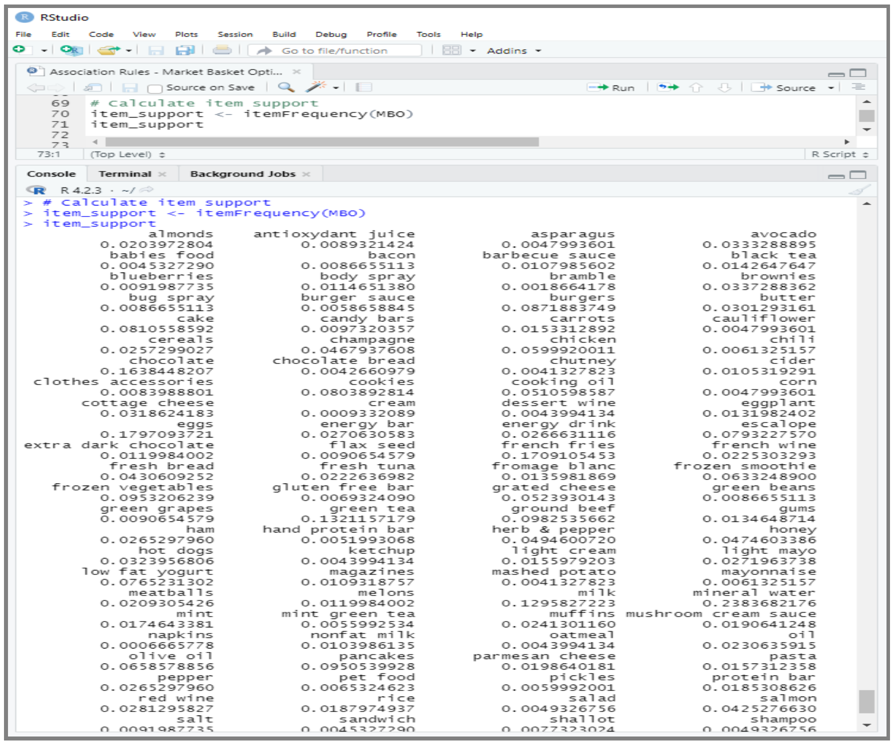
```
# Calculate item support
item_support <- itemFrequency(MBO)
item_support
```

This provides the results shown below. The values represent the item support for each item in the dataset. Item support measures how frequently an item appears in the transactions relative to the total number of transactions.

Here's what each part of the output means:

- **Item Name**: This column lists the items for which support is calculated.
- **Support Value**: This column contains the support values for each item. The support value of an item indicates the proportion of transactions in which that item appears. For example, if the support value for "almonds" is 0.0204, it means that almonds appear in approximately 2.04% of all transactions in the dataset.

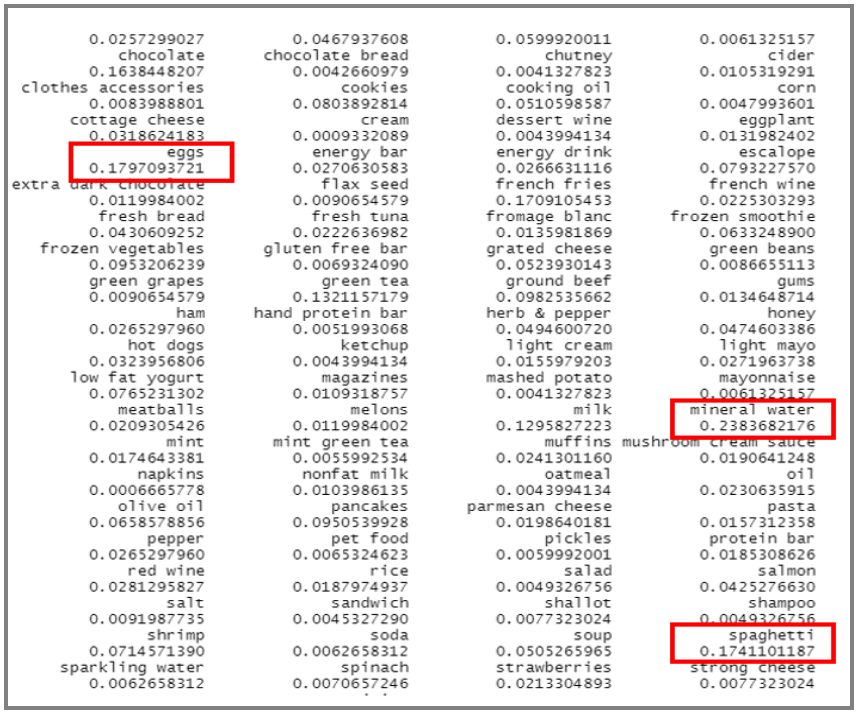
Higher support values suggest that the corresponding items are more frequently purchased or occur together in transactions. These values are essential for identifying frequent itemsets, which are crucial for generating association rules in market basket analysis.



```r
# Calculate item support
item_support <- itemFrequency(MBO)
item_support
```

```
> # Calculate item support
> item_support <- itemFrequency(MBO)
> item_support
             almonds    antioxydant juice            asparagus              avocado
        0.0203972804         0.0089321424         0.0047993601         0.0333288895
         babies food                bacon       barbecue sauce            black tea
        0.0045327290         0.0086655113         0.0107985602         0.0142647647
         blueberries           body spray              bramble             brownies
        0.0091987735         0.0114651380         0.0018664178         0.0337288362
           bug spray         burger sauce              burgers               butter
        0.0086655113         0.0058658845         0.0871883749         0.0301293161
                cake           candy bars              carrots          cauliflower
        0.0810558592         0.0097320357         0.0153312892         0.0047993601
             cereals            champagne              chicken                chili
        0.0257299027         0.0467937608         0.0599920011         0.0061325157
           chocolate      chocolate bread              chutney                cider
        0.1638448207         0.0042660979         0.0041327823         0.0105319291
 clothes accessories              cookies          cooking oil                 corn
        0.0083988801         0.0803892814         0.0510598587         0.0047993601
      cottage cheese                cream         dessert wine             eggplant
        0.0318624183         0.0009332089         0.0043994134         0.0131982402
                eggs           energy bar         energy drink             escalope
        0.1797093721         0.0270630583         0.0266631116         0.0793227570
extra dark chocolate            flax seed         french fries          french wine
        0.0119984002         0.0090654579         0.1709105453         0.0225303293
         fresh bread           fresh tuna        fromage blanc      frozen smoothie
        0.0430609252         0.0222636982         0.0135981869         0.0633248900
   frozen vegetables      gluten free bar        grated cheese          green beans
        0.0953206239         0.0069324090         0.0523930143         0.0086655113
        green grapes            green tea          ground beef                 gums
        0.0090654579         0.1321157179         0.0982535662         0.0134648714
                 ham     hand protein bar        herb & pepper                honey
        0.0265297960         0.0051993068         0.0494600720         0.0474603386
            hot dogs              ketchup          light cream           light mayo
        0.0323956806         0.0043994134         0.0155979203         0.0271963738
      low fat yogurt            magazines        mashed potato           mayonnaise
        0.0765231302         0.0109318757         0.0041327823         0.0061325157
           meatballs               melons                 milk        mineral water
        0.0209305426         0.0119984002         0.1295827223         0.2383682176
                mint       mint green tea              muffins mushroom cream sauce
        0.0174643381         0.0055992534         0.0241301160         0.0190641248
             napkins          nonfat milk              oatmeal                  oil
        0.0006665778         0.0103986135         0.0043994134         0.0230635915
           olive oil             pancakes      parmesan cheese                pasta
        0.0658578856         0.0950539928         0.0198640181         0.0157312358
              pepper             pet food              pickles          protein bar
        0.0265297960         0.0065324623         0.0059992001         0.0185308626
            red wine                 rice                salad               salmon
        0.0281295827         0.0187974937         0.0049326756         0.0425276630
                salt             sandwich              shallot              shampoo
        0.0091987735         0.0045327290         0.0077323024         0.0049326756
```

Recall that the top three (3) most frequently purchased items were mineral water, eggs, and spaghetti. The item support also indicates the top three (3).



```
        0.0257299027         0.0467937608         0.0599920011         0.0061325157
           chocolate      chocolate bread              chutney                cider
        0.1638448207         0.0042660979         0.0041327823         0.0105319291
 clothes accessories              cookies          cooking oil                 corn
        0.0083988801         0.0803892814         0.0510598587         0.0047993601
      cottage cheese                cream         dessert wine             eggplant
        0.0318624183         0.0009332089         0.0043994134         0.0131982402
                eggs           energy bar         energy drink             escalope
        0.1797093721         0.0270630583         0.0266631116         0.0793227570
extra dark chocolate            flax seed         french fries          french wine
        0.0119984002         0.0090654579         0.1709105453         0.0225303293
         fresh bread           fresh tuna        fromage blanc      frozen smoothie
        0.0430609252         0.0222636982         0.0135981869         0.0633248900
   frozen vegetables      gluten free bar        grated cheese          green beans
        0.0953206239         0.0069324090         0.0523930143         0.0086655113
        green grapes            green tea          ground beef                 gums
        0.0090654579         0.1321157179         0.0982535662         0.0134648714
                 ham     hand protein bar        herb & pepper                honey
        0.0265297960         0.0051993068         0.0494600720         0.0474603386
            hot dogs              ketchup          light cream           light mayo
        0.0323956806         0.0043994134         0.0155979203         0.0271963738
      low fat yogurt            magazines        mashed potato           mayonnaise
        0.0765231302         0.0109318757         0.0041327823         0.0061325157
           meatballs               melons                 milk        mineral water
        0.0209305426         0.0119984002         0.1295827223         0.2383682176
                mint       mint green tea              muffins mushroom cream sauce
        0.0174643381         0.0055992534         0.0241301160         0.0190641248
             napkins          nonfat milk              oatmeal                  oil
        0.0006665778         0.0103986135         0.0043994134         0.0230635915
           olive oil             pancakes      parmesan cheese                pasta
        0.0658578856         0.0950539928         0.0198640181         0.0157312358
              pepper             pet food              pickles          protein bar
        0.0265297960         0.0065324623         0.0059992001         0.0185308626
            red wine                 rice                salad               salmon
        0.0281295827         0.0187974937         0.0049326756         0.0425276630
                salt             sandwich              shallot              shampoo
        0.0091987735         0.0045327290         0.0077323024         0.0049326756
              shrimp                 soda                 soup            spaghetti
        0.0714571390         0.0062658312         0.0505265965         0.1741101187
     sparkling water              spinach         strawberries        strong cheese
        0.0062658312         0.0070657246         0.0213304893         0.0077323024
```

The output is provided alphabetically, which can be helpful if I am looking for a specific item. However, I can also sort the results in descending order of item support using the following code:

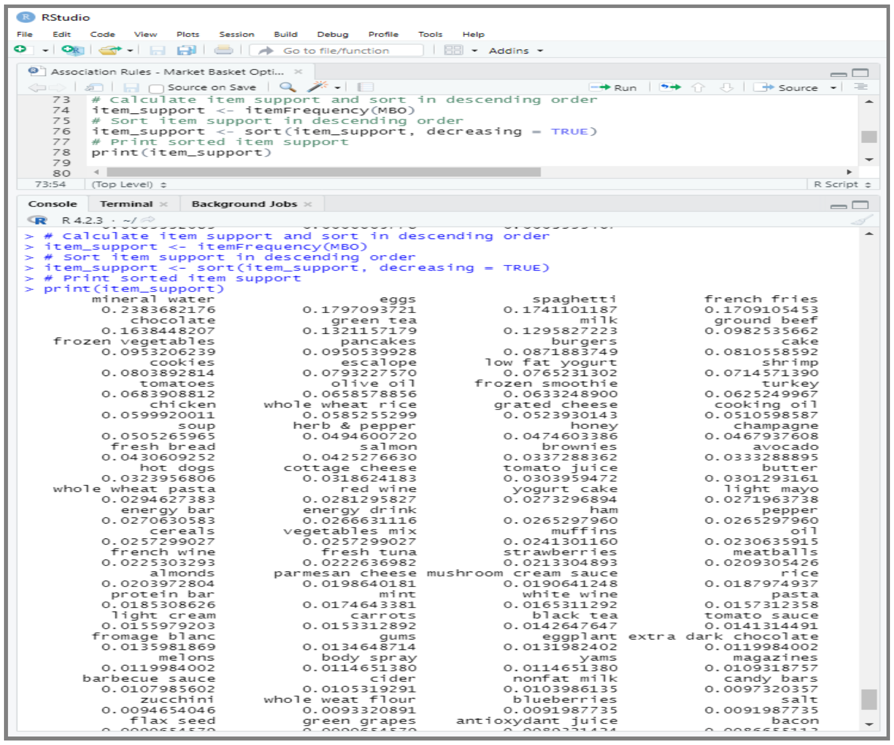
```
# Calculate item support and sort in descending order
item_support <- itemFrequency(MBO)
# Sort item support in descending order
item_support <- sort(item_support, decreasing = TRUE)
# Print sorted item support
print(item_support)
```

This provides the results below. These results provide information about the item support and the proportion of transactions in the dataset containing each item. Each item is listed along with its corresponding support value, sorted in descending order.

For example:

- "Mineral water" has a support of approximately 23.84%, meaning it appears in about 23.84% of all transactions.
- "Eggs" have a support of approximately 17.97%, meaning they appear in about 17.97% of all transactions.
- "Spaghetti" has a support of approximately 17.41%, meaning it appears in about 17.41% of all transactions.
- And so on for each item listed.

Here, we can quickly see that the top 10 items match our previous findings. These support values are crucial for association rule mining as they help identify frequent itemsets that occur together frequently in transactions.



```
# Calculate item support and sort in descending order
item_support <- itemFrequency(MBO)
# Sort item support in descending order
item_support <- sort(item_support, decreasing = TRUE)
# Print sorted item support
print(item_support)
```

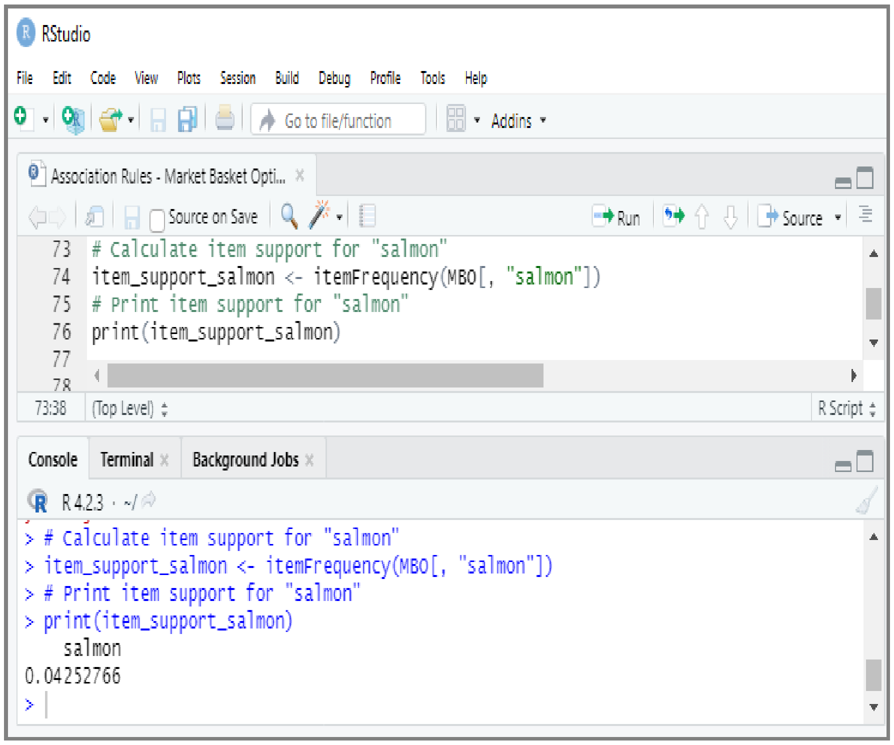
```
> # Calculate item support and sort in descending order
> item_support <- itemFrequency(MBO)
> # Sort item support in descending order
> item_support <- sort(item_support, decreasing = TRUE)
> # Print sorted item support
> print(item_support)
       mineral water                eggs           spaghetti        french fries
        0.2383682176         0.1797093721         0.1741101187         0.1709105453
           chocolate            green tea                milk         ground beef
        0.1638448207         0.1321157179         0.1295827223         0.0982535662
   frozen vegetables             pancakes              burgers                cake
        0.0953206239         0.0950539928         0.0871883749         0.0810558592
             cookies             escalope       low fat yogurt              shrimp
        0.0803892814         0.0793227570         0.0765231302         0.0714571390
            tomatoes            olive oil      frozen smoothie              turkey
        0.0683908812         0.0658578856         0.0633248900         0.0625249967
             chicken     whole wheat rice        grated cheese         cooking oil
        0.0599920011         0.0585255299         0.0523930143         0.0510598587
                soup        herb & pepper                honey           champagne
        0.0505265965         0.0494600720         0.0474603386         0.0467937608
         fresh bread               salmon             brownies             avocado
        0.0430609252         0.0425276630         0.0337288362         0.0333288895
            hot dogs       cottage cheese         tomato juice              butter
        0.0323956806         0.0318624183         0.0303959472         0.0301293161
   whole wheat pasta             red wine          yogurt cake          light mayo
        0.0294627383         0.0281295827         0.0273296894         0.0271963738
          energy bar         energy drink                  ham              pepper
        0.0270630583         0.0266631116         0.0265297960         0.0265297960
             cereals       vegetables mix              muffins                 oil
        0.0257299027         0.0257299027         0.0241301160         0.0230635915
         french wine           fresh tuna         strawberries           meatballs
        0.0225303293         0.0222636982         0.0213304893         0.0209305426
             almonds      parmesan cheese mushroom cream sauce                rice
        0.0203972804         0.0198640181         0.0190641248         0.0187974937
         protein bar                 mint           white wine               pasta
        0.0185308626         0.0174643381         0.0165311292         0.0157312358
         light cream              carrots            black tea        tomato sauce
        0.0155979203         0.0153312892         0.0142647647         0.0141314491
       fromage blanc                 gums             eggplant extra dark chocolate
        0.0135981869         0.0134648714         0.0131982402         0.0119984002
              melons           body spray                 yams           magazines
        0.0119984002         0.0114651380         0.0114651380         0.0109318757
      barbecue sauce                cider          nonfat milk          candy bars
        0.0107985602         0.0105319291         0.0103986135         0.0097320357
            zucchini     whole weat flour          blueberries                salt
        0.0094654046         0.0093320891         0.0091987735         0.0091987735
           flax seed         green grapes    antioxydant juice               bacon
```

If we would like to find the item support for a particular item, such as salmon, we can use the following code:

```
# Calculate item support for "salmon"
item_support_salmon <- itemFrequency(MBO[, "salmon"])
# Print item support for "salmon"
print(item_support_salmon)
```

This provides the item support shown below, which indicates that "salmon" appears in approximately 4.25% of all transactions in the dataset.

Understanding the item support for a specific item like "salmon" is crucial in association rule mining. It helps identify how frequently "salmon" is purchased relative to the total number of transactions. Higher support values suggest that the item is popular among customers and may have significant associations with other items in the dataset.



```
# Calculate item support for "salmon"
item_support_salmon <- itemFrequency(MBO[, "salmon"])
# Print item support for "salmon"
print(item_support_salmon)
```

```
> # Calculate item support for "salmon"
> item_support_salmon <- itemFrequency(MBO[, "salmon"])
> # Print item support for "salmon"
> print(item_support_salmon)
    salmon
0.04252766
>
```

Build the Model

After calculating item support, our next step is to fit the model and train the Apriori algorithm on the dataset. This next step aims to uncover significant relationships between variables in large datasets. The method relies on the Apriori algorithm to identify frequent itemsets from transaction data and subsequently generate association rules based on these itemsets.

Recall that association rules typically form "if-then" statements, providing insights into the co-occurrence patterns of items within transactions. For instance, an association rule might suggest that customers who purchase milk and bread are also likely to buy butter.

A key aspect of the Apriori algorithm is its pruning technique, which leverages the "apriori property" to streamline the search for frequent itemsets. This property dictates that if an itemset is frequent, then all of its subsets must also be frequent. By exploiting this property, Apriori can efficiently navigate the vast search space of possible itemsets, improving computational efficiency.

We will use the Apriori algorithm for Association Rule mining using the following code:

```
# Fit the model
# Train Apriori on the dataset
set.seed = 220 # Setting seed
association_rules <- apriori(data = MBO, parameter =
list(support = 0.002, confidence = 0.1))
```

Let's break down the code:

- `set.seed = 220`: This line sets the seed for the random number generator. Setting the seed ensures reproducibility of the results, meaning that if you run the code multiple times with the same seed, you should get the same results.

- `association_rules <- apriori(data = MBO, parameter = list(support = 0.002, confidence = 0.1))`: This line trains the Apriori algorithm on the dataset `MBO` to discover association rules. Here's a breakdown of the parameters used:
    - `data`: Specifies the dataset on which the algorithm will operate. In this case, `MBO` is the transaction dataset.

    - `parameter`: This parameter allows you to specify various settings for the Apriori algorithm. In this example, a list of parameters is provided:
        - `support = 0.002`: Sets the minimum support threshold to 0.002. Support refers to the proportion of transactions in the dataset that contain the itemset. Setting a low support threshold allows the algorithm to consider only those itemsets that appear in a minimum proportion of transactions.

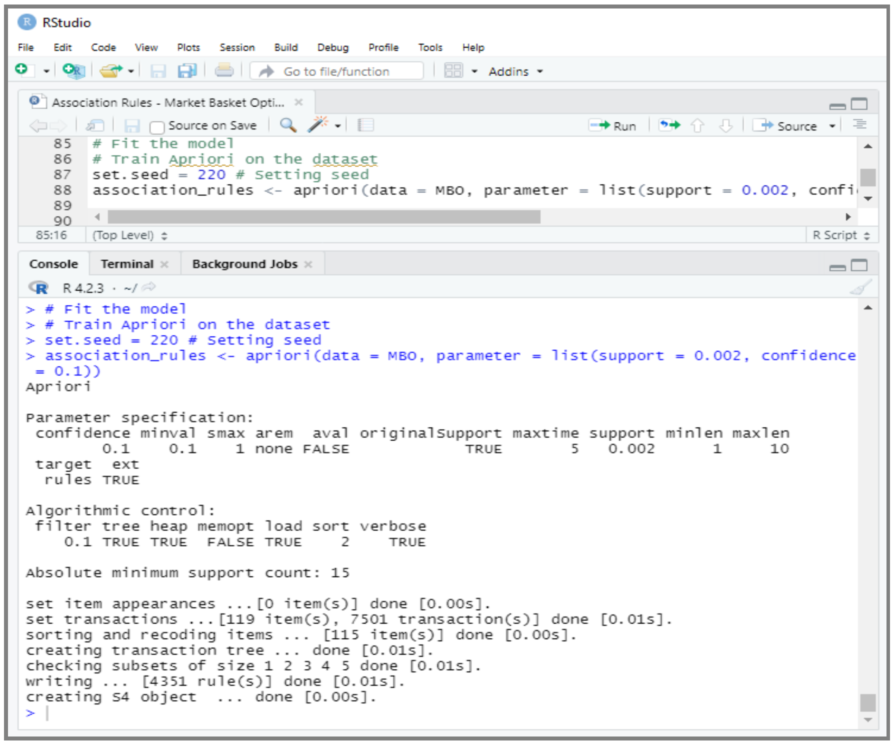
        - `confidence = 0.1`: Sets the minimum confidence threshold to 0.1. Confidence measures the reliability of the association rule. It is the proportion of transactions containing the antecedent (left-hand side of the rule) where the consequent (right-hand side) is also present. A confidence of 0.1 means that the rule must be correct at least 10% of the time for it to be considered significant.

Overall, this code initializes the Apriori algorithm with specific parameters for support and confidence thresholds and trains it on the transaction dataset to discover association rules that meet these criteria. The resulting association rules will provide insights into the relationships between different items purchased together in transactions.

This code provides the results below. Let's break down the important parts of the output:

- **Parameter Specification**: This section outlines the parameters used for the Apriori algorithm:
    - `confidence`: Minimum confidence threshold set to 0.1, indicating the minimum confidence level required for a rule to be considered significant.
    - `support`: The minimum support threshold is set to 0.002, representing the minimum proportion of transactions containing a particular itemset to be considered frequent.
    - `minlen` and `maxlen`: Minimum and maximum lengths of the rules generated. In this case, rules can have between 1 and 10 items.
    - `target`: Indicates that the objective is to generate association rules.
- **Algorithmic Control**: This section specifies various control parameters for the algorithm:
    - `filter`, `tree`, `heap`, `memopt`, `load`, `sort`, `verbose`: Control flags for specific optimizations and settings during the algorithm's execution.
- **Absolute Minimum Support Count**: The minimum support count is set to 15. This value represents the absolute minimum number of transactions in which an itemset must appear to be considered frequent.
- **Data Processing Steps**:
    - `set item appearances`: This step indicates whether specific items were pre-specified to appear in transactions (which is not the case here).
    - `set transactions`: The dataset contains 119 unique items and 7501 transactions.
    - `sorting and recoding items`: Items are sorted and recoded for efficient processing.
    - `creating transaction tree`: A data structure is created to store and process transactions efficiently.
    - `checking subsets of size`: Subsets of different sizes are checked during rule generation.

- `writing`: The generated rules are written.
- `creating S4 object`: Finally, the result is stored as an S4 object, which is a complex data structure used in R for representing objects with multiple components.



```
# Fit the model
# Train Apriori on the dataset
set.seed = 220 # Setting seed
association_rules <- apriori(data = MBO, parameter = list(support = 0.002, confi
```

```
> # Fit the model
> # Train Apriori on the dataset
> set.seed = 220 # Setting seed
> association_rules <- apriori(data = MBO, parameter = list(support = 0.002, confidence
 = 0.1))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen
        0.1    0.1    1 none FALSE            TRUE       5   0.002      1     10
 target  ext
  rules TRUE

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

Absolute minimum support count: 15

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.01s].
sorting and recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.01s].
checking subsets of size 1 2 3 4 5 done [0.01s].
writing ... [4351 rule(s)] done [0.01s].
creating S4 object  ... done [0.00s].
> 
```

So what association rules were generated? We can check this by running the following code:

```
association_rules
```

This code provides the output below. The output "set of 4351 rules" indicates that the `association_rules` object contains a collection of 4,351 association rules generated by the Apriori algorithm during mining. These rules represent the discovered relationships between items in the dataset based on the specified support and confidence thresholds.

Each rule in the set typically consists of three components:

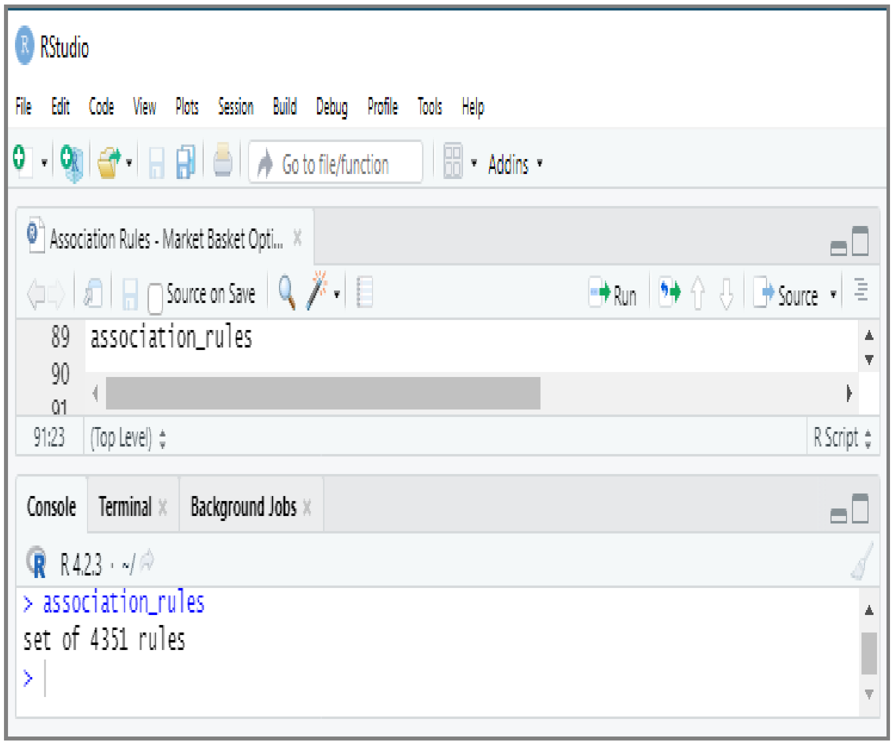
- **Antecedent**: The items or itemsets that appear in the "if" part of the rule, representing the conditions or circumstances under which the consequent is expected to occur.
- **Consequent**: The items or itemsets that appear in the "then" part of the rule, representing the outcomes or events that are predicted to occur given the presence of the antecedent.
- **Metrics**: Associated measures such as support, confidence, lift, or other metrics that quantify the strength or significance of the rule.

For example, a rule might be expressed as follows:

"If {milk, bread} then {eggs}" with a support of 0.05 and a confidence of 0.8.

This rule suggests that customers who purchase milk and bread together have an 80% likelihood of also buying eggs, and this combination occurs in 5% of all transactions.

In our case, the set of 4,351 rules indicates that the Apriori algorithm has identified and generated 4,351 such rules based on the specified parameters and the transaction data provided. These rules can be further analyzed, evaluated, and interpreted to derive actionable insights for various applications such as market basket analysis, product recommendations, or cross-selling strategies.



```
association_rules
```

```
> association_rules
set of 4351 rules
>
```

Let's say we want to filter the association rules related to mineral water since this was the most frequently purchased item. We can filter the association rules for mineral water using the following code:

```
# Filter association rules for "mineral water"
mineral_water_rules <- subset(association_rules,
                              subset = lhs %in% "mineral water" | rhs %in%
"mineral water")
# View the rules related to "mineral water"
print(mineral_water_rules)
inspect(mineral_water_rules)
```

This filters the association rules down to 1,441, which is too many rules for this example. So, let's find an item with a smaller item support. Chili has an item support value of 0.0061325157. It's not the lowest, but it falls about two-thirds of the way down.

We can filter the association rules for chili using the following code:

```
# Filter association rules for "chili"
chili_rules <- subset(association_rules,
                      subset = lhs %in% "chili" | rhs %in% "chili")
# View the rules related to "chili"
print(chili_rules)
inspect(chili_rules)
```

This provides the output below. The output provides details about association rules involving the item "chili". Here's what each column means:

- **lhs**: The left-hand side (lhs) of the association rule represents the antecedent (the items present before the arrow).
- **rhs**: The right-hand side (rhs) of the association rule, representing the consequent (the items that are predicted or appear after the arrow).
- **support**: The support of the rule indicates the proportion of transactions that contain both the lhs and the rhs.
- **confidence**: The confidence of the rule, which suggests the probability of finding the rhs in a transaction given that it contains the lhs.
- **coverage**: The coverage of the rule indicates the proportion of transactions, including the lhs.
- **lift**: The lift of the rule measures how much more likely the rhs is in transactions containing the lhs than its overall likelihood.
- **count**: The absolute count of transactions that satisfy the rule.

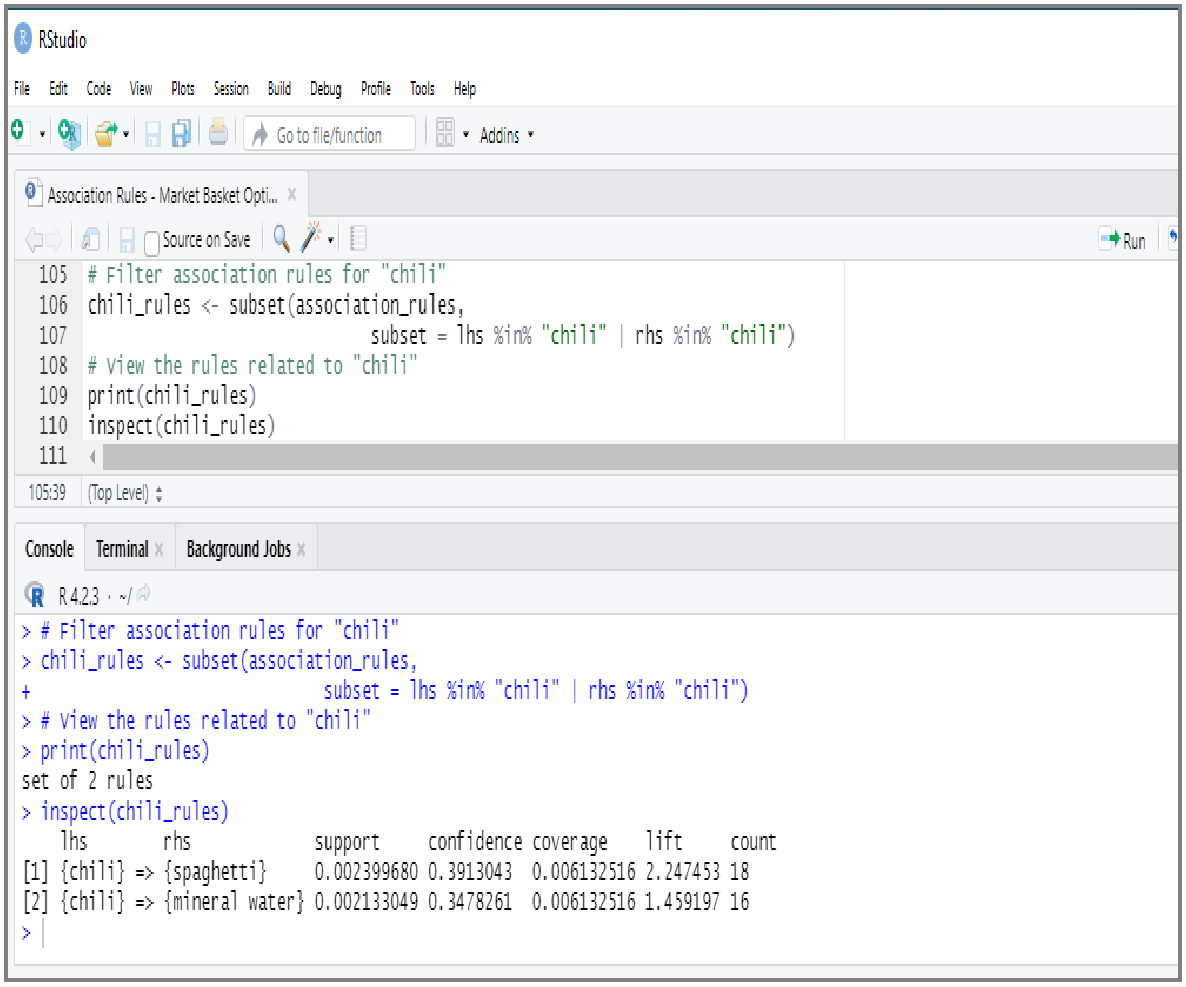
In this specific output, there are two association rules involving "chili":

- If "chili" is purchased, there's a 0.24% chance (support) it will also include "spaghetti". The confidence is 39.13%, meaning that in 39.13% of transactions, "chili" and "spaghetti" are also present. The lift of 2.25

indicates that "spaghetti" is 2.25 times more likely to be bought when "chili" is bought compared to its general likelihood.

- If "chili" is purchased, there's a 0.21% chance (support) it will also include "mineral water". The confidence is 34.78%, indicating that in 34.78% of transactions containing "chili", "mineral water" is also present. The lift of 1.46 suggests that "mineral water" is 1.46 times more likely to be bought when "chili" is bought than its general likelihood.

These rules provide insights into potential associations between "chili" and other items in the dataset.



```
# Filter association rules for "chili"
chili_rules <- subset(association_rules,
                      subset = lhs %in% "chili" | rhs %in% "chili")
# View the rules related to "chili"
print(chili_rules)
inspect(chili_rules)
```

```
> # Filter association rules for "chili"
> chili_rules <- subset(association_rules,
+                       subset = lhs %in% "chili" | rhs %in% "chili")
> # View the rules related to "chili"
> print(chili_rules)
set of 2 rules
> inspect(chili_rules)
    lhs        rhs             support     confidence coverage    lift     count
[1] {chili} => {spaghetti}     0.002399680 0.3913043  0.006132516 2.247453 18
[2] {chili} => {mineral water} 0.002133049 0.3478261  0.006132516 1.459197 16
> 
```

Now that we have created our association rules, our next step is to evaluate their performance.

In the last step, we used Association Rules to provide insights into item associations. Now, we must evaluate the performance of our model. Please continue to follow along using the dataset.

Market_Basket_Optimisation.csv

Evaluation

This step assesses the models' performance. This step determines if the association rules meet the predefined criteria and produce reliable results aligned with the business objectives. Ask yourself if the model or models you choose will have the desired results.

Evaluate Association Rules Effectiveness and Strength

It is essential to evaluate the effectiveness or strength of the Association Rules. Since this is an Association Rules problem, we will focus on confidence and lift. Note that we already evaluated support. Let's review these again briefly:

- **Support:** Support measures the frequency of occurrence of an itemset in the dataset. It is calculated as the proportion of transactions in the dataset that contain the itemset.
- **Confidence:** Confidence measures the reliability of an association rule. It indicates the likelihood that the consequent will occur given that the antecedent has occurred.
- **Lift:** Lift measures the strength of association between the antecedent and consequent in an association rule relative to their individual support levels.

Collectively, these elements form the foundation for a comprehensive evaluation of the Association Rules.

- **Confidence**
- **Lift**

Create a Graphical Representation of the Association Rules

Finally, we can plot the association rules to create a graphical representation of the association rules, allowing analysts to visually explore and interpret the relationships between different items or itemsets in the dataset. This visualization can provide valuable insights into patterns of co-occurrence and association strength, aiding in decision-making processes such as product placement, cross-selling strategies, or customer segmentation.

We can plot the association rules using the following code:

```
# Plot the association rules
plot(association_rules, method = "graph",
    measure = "confidence", shading = "lift", control = list(max =
50))
```

Here's a breakdown of each parameter:

- `association_rules`: This is the input data containing the association rules. These rules are typically generated through algorithms like

Apriori or FP-Growth, which analyze transactional data to identify co-occurrence patterns among items.

- `method = "graph"`: Specifies the method used for plotting the association rules. In this case, it creates a graph-based visualization.
- `measure = "confidence"`: Determines how the rules will be displayed in the plot. Confidence is a common measure used in association rule mining that represents the conditional probability of the consequent given the antecedent.
- `shading = "lift"`: Specifies how the shading of the plot will be determined. In this case, it uses the lift measure to determine the shading. Lift is another measure used in association rule mining that indicates the strength of association between antecedent and consequent. Shading the plot based on lift can help visually highlight stronger associations.
- `control = list(max = 50)`: Provides control parameters for the plot. In this case, it limits the maximum number of rules to be plotted to 50. This is useful for managing the complexity of the visualization, especially when dealing with a large number of association rules.

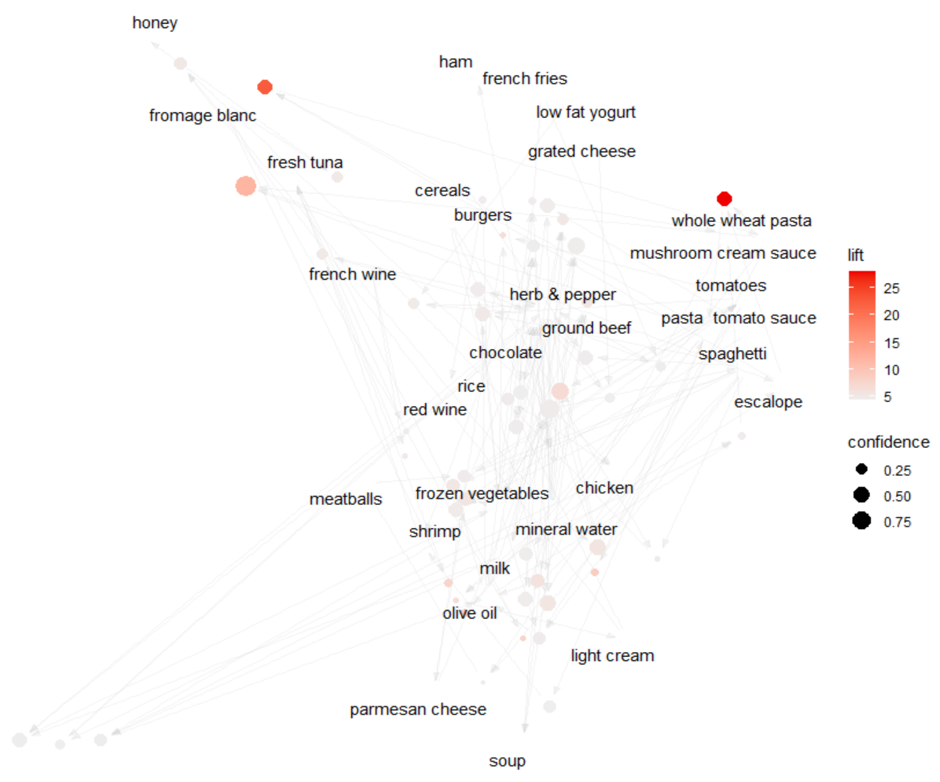
This provides the plot below. Let's take a look at the lift and confidence values.

In the context of association rule visualization, using the `plot` function with the `shading = "lift"` parameter set to determine shading, darker red typically indicates higher lift values. Shading based on lift helps visually highlight rules with stronger associations when plotting association rules. Darker shades of red suggest higher lift values, indicating stronger relationships between the items in the antecedent and consequent of the rule. This can be particularly useful for identifying significant patterns or associations in the dataset, allowing analysts to focus on the most relevant rules for further analysis or decision-making.

If we look at the sorted lift values, the highest lift value is 28.088096. This value is associated with escalope and mushroom cream sauce on the left-hand side (lhs) and pasta on the right-hand side (rhs).

Let's look at confidence next. In the context of association rule visualization, using the `plot` function with the `measure = "confidence"` parameter set, the bigger circles represent association rules with higher confidence values. When plotting association rules, using a circle size to represent confidence allows for easy visual identification of rules with higher confidence levels. Larger circles indicate higher confidence, suggesting a stronger likelihood that the consequent will occur when the antecedent is present. This visual representation helps users quickly discern which rules have stronger conditional probabilities, aiding decision-making or further analysis of the association rules.

If we look at the sorted confidence values, the highest confidence value is 0.9500000. This value is associated with escalope and mushroom cream sauce on the left-hand side (lhs) and pasta on the right-hand side (rhs).


honey
ham
french fries
fromage blanc
low fat yogurt
grated cheese
fresh tuna
cereals
burgers
whole wheat pasta
mushroom cream sauce
french wine
tomatoes
herb & pepper
pasta
tomato sauce
ground beef
spaghetti
chocolate
rice
escalope
red wine
meatballs
frozen vegetables
chicken
shrimp
mineral water
milk
olive oil
light cream
parmesan cheese
soup
lift
confidence