Table of Contents

List of Figures

Chapter 1: Introduction

1.1. Introduction

Association analysis is the process of uncovering interesting correlations hidden within massive datasets. These interesting correlations can present themselves in one of two ways: frequent item sets or association rules. A group of items that commonly appear together is referred to as a frequent item set. Association rules are the second perspective through which one can observe intriguing interactions. Rules of association imply that there is a close connection between the two things being analyzed. Market Basket Analysis is a strategy utilised by businesses to boost sales by gaining an understanding of client purchasing habits and behaviour. In a business context, market basket analysis studies collections of items to determine their relationships and correlations. It identifies combinations of items that frequently occur together in transactions.

1.2. Problem Statement

Given that there are a great number of retailers that offer a variety of goods and services to the customers in today's competitive world, the placement of items and bundling offering are some of the challenges that many different types of retailers or shop owners face. Since they are not aware of the shopping preferences of the customer, they are unable to determine which products should be grouped together in their store. With the strong competition with retailers offering the same products and better deals, many retailers face stagnant sales due to lack of strategy.

The purpose of this assignment is to develop an application whereby retailers are able to determine the strong relationships that exist between the products with the assistance of this application, which ultimately enables them to place products that frequently occur together in close proximity to one another. Also, the insights obtained can facilitate decisions on which item to stock more of, cross selling, up selling, and the arrangement of store shelves.

1.3. Objectives

1. To develop a Python command line program that allows users to upload or type records, implements Apriori and FP growth modules, and perform association rules on frequent itemsets.
2. To deliver a comprehensive report detailing the design and development of association analysis program.
3. To help retailers make decisions on strategic product placement, offer special deals and create new, attractive product bundles based on customers purchase patterns in order to boost sales.

1.4. Scope

The target users are **retailers**. A retailer sells goods to the public in relatively small quantities for use or consumption rather than for resale. Market Basket Analysis is a method used by retailers to uncover the associations that exist between different products. This is done by looking for items that are frequently purchased together in transactions and analyse how strong they are associated. The strategy is predicated on the assumption that clients who purchase one item are more likely to purchase another item from the same category in the future.

This assignment consists of two parts. Firstly, a program developed using Python that mines frequent patterns using either Apriori or FP growth module from user-generated inputs or file upload. Then, the program constructs the association rules based on the resulting frequent patterns. Throughout the program, users shall be allowed to input threshold values such as minimum support count, confidence values, and lift. Next, a comprehensive report detailing the background study, data preprocessing on case study datasets, design and implementation of program, stages of output, and discussion on the findings made.

Chapter 2: Literature Review

2.1. Literature on the transactional data.

Transactional data is any information that is collected as a result of a business event. It is simple to think that a transaction must involve a purchase such as an online purchase. The fact that transactional data includes a date and time distinguishes it from other types of data collection. When evaluating customer behaviours, this temporal detail offers it an objective piece of data to use as a pivot point and helps to provide context. When analysed properly, transactional data can become incredibly valuable. It offers context to routine activities and enables businesses to better comprehend what their customers are doing online. Transactional data allows businesses to track their customers' clicks and behaviour before and after a purchase. It can even assist in determining why customers leave a website without making a purchase or abandon their shopping carts (Castillo, 2022).

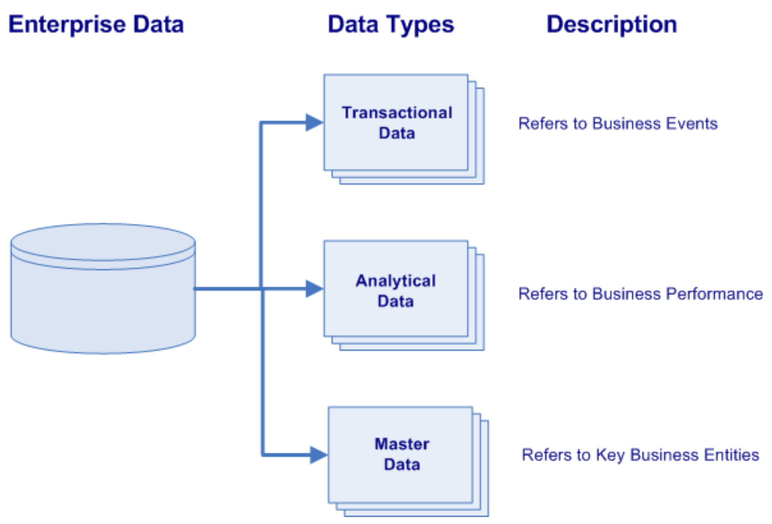

Enterprise Data
Data Types
Description
Transactional Data
Refers to Business Events
Analytical Data
Refers to Business Performance
Master Data
Refers to Key Business Entities

Figure 2.1: Types of Enterprise Data

Figure above depicts the three main data types of enterprise data, namely transactional data, analytical data, and master data (Getz, 2011). Transactional data support an organization's present operations and are integrated into application systems that automate core business processes. This could include sales, service, order management, production, purchasing, billing, and accounts receivable and payable. Transactional data often refers to data created and updated within operational systems. The time, location, price, discount, and payment methods utilised at the point of sale are examples of transactional data. Online Transaction Processing (OLTP) systems frequently store transactional data in normalised, data-integrity-focused tables. Rather of being the subjects of a transaction, such as a customer or a product, transactional data comprises of identifying data, such as time and numeric values.

Analytical data, in contrast, consists of numerical values, metrics, and indicators that provide corporate insight and improve organisational decision-making. Online Analytical Processing (OLAP) repositories optimised for decision support, such as enterprise data warehouses and department data marts, are frequently used to store analytical data. In a dimensional model, analytical data is represented by facts and numeric values. Typically, data exists in tables surrounded by key dimensions such as customer, product, account, location, and date/time. In contrast to descriptive data, however, analytical data are characterised by numerical measures.

On the other hand, master data is an integral part of an organization's activities. Master data refers to the core organisational entities utilised by various functional units and frequently stored in multiple data systems within an organisation. In addition, master data represents the business entities that the organization's business transactions revolve around, as well as the organization's primary analytical elements. Master data are frequently non-transactional, durable data that are utilised by several systems to define the primary business entities. Customer, product, employee, inventory, supplier, and location information are examples of master data.

Transactional data gives a benefit for optimising business operations. It is beneficial to a business for both preventative maintenance and boosting operational procedures. Insights derived from transactional data are eventually intuitive and may be utilised to offer greater customer experiences.

2.2. Literature on the frequent pattern mining techniques.

The analysis of data is crucial to the decision-making process. This type of pattern analysis delivers various benefits, including increased revenue, decreased expenses, and a strengthened competitive edge. Mining the hidden patterns of regularly occurring item sets gets more time-consuming as the amount of data increases. In addition, the algorithm's intense computations need a large amount of memory for mining the hidden patterns of frequent item sets. Consequently, an efficient algorithm is required to mine the hidden patterns of the frequent item sets in a shorter amount of time and with less memory consumption as the volume of data increases. (Chee, Jaafar, Aziz, Hasan, & Yeoh, 2018).

Numerous scholars have proposed different strategies to improve the Frequent Pattern Mining (FPM) methodology. However, performance enhancements of existing FPM algorithms are still required because the majority of present algorithms are not suited for mining a massive data set with an increasing amount of data. The two greatest obstacles faced by the majority of FPM algorithms are lengthy execution time and excessive memory consumption when mining all the hidden frequent patterns (Jamsheela & Raju, 2015).

2.2.1. Apriori Algorithm

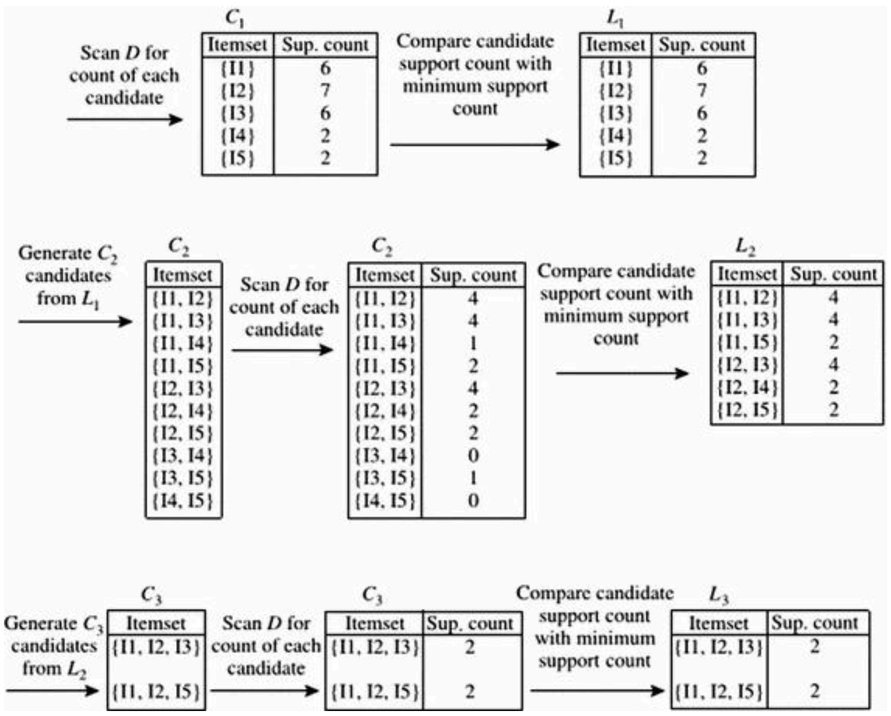
The two FPM algorithms in the context of this assignment are Apriori algorithm and Frequent Pattern Growth (FP-Growth). Firstly, Apriori algorithm takes a level-wise approach that generates all frequently occurring itemsets of length k before those of length (k + 1). Every subset of a frequent pattern is also frequent, which is the fundamental concept that the Apriori method is based on (Aggarwal, Bhuiyan, & Hasan, 2014).

```
Algorithm1: Apriori algorithm
Input:   D: Input Dataset
         minSup: minimum support threshold
Output:  All 2 to k-frequent itemsets

 1. $L_1$= {1-frequent itemset} // found separately
 2. for ($k$ = 2; $L_{k-1} \neq \varphi$; $k$++)
 3.   $C_k$ = apriori_gen($L_{k-1}$) // finds k-candidate itemsets by
        joining and pruning $L_{k-1}$ with itself
 4.     for each transaction t in D
 5.         $C_t$ = subset ($C_k$, t) // finds candidate itemsets in t
 6.         for each c in $C_t$
 7.             c.count++
 8.         end for each
 9.     end for each
10.     $L_k = \{c \in C_k \mid c.\text{count} \geq minSup\}$
11. end for
12. Return $\cup_k L_k$
```

Figure 2.2: Apriori Algorithm Pseudocode

Figure above shows the pseudocode for running an Apriori algorithm. In the initial iteration, the input dataset is scanned to determine the support of each item. 1-frequent item sets are determined by filtering only those items whose support count is equal to or more than a minimum support threshold value provided by the user (line 1). Then, (k-1) frequent item sets were employed to identify k-frequent item sets (where k > 1). (line 2). The join procedure creates all k-candidate sets by joining (k-1) frequent sets to itself (line 3). The prune phase uses the Apriori property to reduce the number of candidate item sets (lines 4–5) to only the most promising candidates by utilising the Apriori property. The input dataset is then iterated a second time to assess the frequency of these prospective candidate itemsets in order to exclude the k-frequent candidate itemsets (line 6–10). The set of all k-frequent item sets (k1) is then returned as the final result (line 12) (Raj, Ramesh, & Sethi, 2021).


Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan D for count of each candidate
Compare candidate support count with minimum support count

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 2.3: Worked Example for Apriori Algorithm

The Apriori approach dramatically reduces the size of the candidate itemsets while also improving performance. However, it continues to have two drawbacks. First, if the total count of a common k-itemset increases, many candidate itemsets may still need to be developed. The entire database must then be repeatedly scanned, and a large number of candidate items must then be confirmed using the pattern matching technique (Chee, Jaafar, Aziz, Hasan, & Yeoh, 2018).

2.2.2. FP Growth

Next is FP Growth. Based on Figure above, FP-Growth addresses the issue of detecting long common patterns by periodically searching through smaller Conditional FP-Trees. The Conditional Pattern Base is a "sub-database" composed of each prefix path in the FP-Tree that co-occurs with each frequent length-1 item. It is used to build the Conditional FP-Tree and generate all frequent patterns associated with each frequent length-1 item. Thus, the cost of searching for regular patterns is significantly decreased. Nevertheless, creating the FP-Tree is time-consuming if the data collection is substantial (Chee, Jaafar, Aziz, Hasan, & Yeoh, 2018).

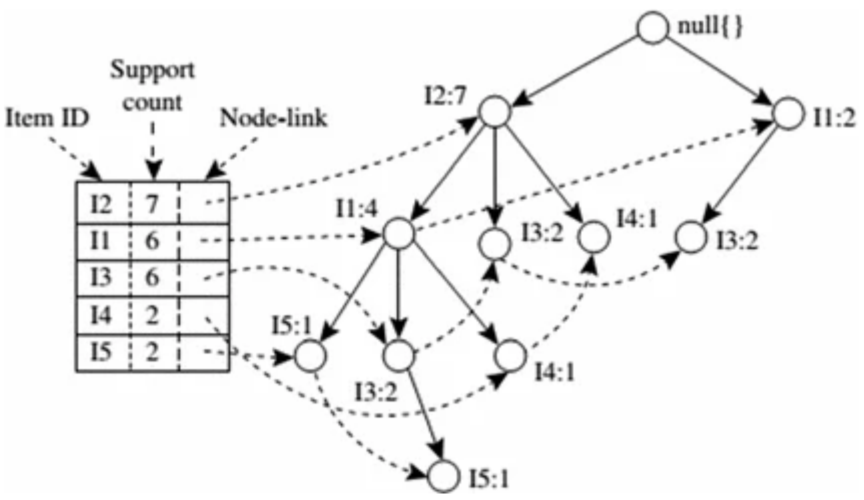
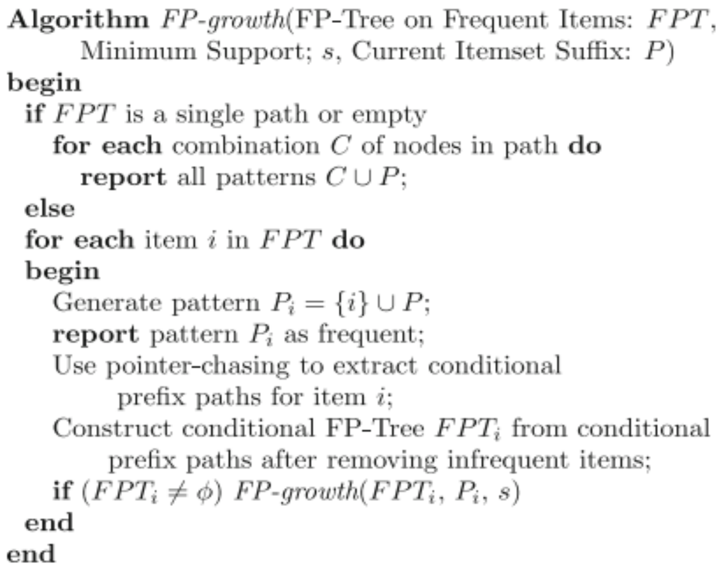


Figure 2.5: FP Tree Visualisation

For more efficient counting, the FP-growth algorithm combines suffix-based pattern analysis with a compressed representation of the projected database. The prefix-based FP-Tree is a compressed representation of the database that is constructed by taking into account a defined order among the items in an itemset. A node in the FP-Tree is labeled with an item, just as each node in a tree is labeled with a symbol. In addition, the node holds the support for the itemset that is defined by the items of the nodes on the path from the root. By combining prefixes, compression is achieved. This is beneficial for memory management. (Aggarwal, Bhuiyan, & Hasan, 2014)



```
Algorithm FP-growth(FP-Tree on Frequent Items: $FPT$,
     Minimum Support; $s$, Current Itemset Suffix: $P$)
begin
  if $FPT$ is a single path or empty
    for each combination $C$ of nodes in path do
      report all patterns $C \cup P$;
  else
  for each item $i$ in $FPT$ do
  begin
    Generate pattern $P_i = \{i\} \cup P$;
    report pattern $P_i$ as frequent;
    Use pointer-chasing to extract conditional
         prefix paths for item $i$;
    Construct conditional FP-Tree $FPT_i$ from conditional
         prefix paths after removing infrequent items;
    if $(FPT_i \neq \phi)$ FP-growth($FPT_i$, $P_i$, $s$)
  end
end
```

Figure 2.4: FP Growth Pseudocode

Figure above depicts the pseudocode for the FP-growth algorithm. The inputs of the algorithm are an FP-Tree FPT, the current itemset suffix P, and user-defined minimum supports. To assist the recursive description, the additional suffix P has been appended to the parameter set P. At the highest-level call performed by the user, P has the value. In addition, the conditional

FP-Tree is generated from a database of often occurring items as opposed to all objects. This attribute is preserved throughout multiple recursive calls. For each item I in an FP-Tree FPT, the conditional FP-Trees are constructed (which is already known to be frequent). The construction of conditional FP-Trees involves chasing pointers for each item in the FP-Tree. This returns all of the item's conditional prefix pathways. The rare nodes on these paths are eliminated, and the remaining nodes are combined to form a conditional FP-Tree FPTi. Due to the removal of infrequent items from FPTi, the new conditional FP-Tree comprises only frequent things. Therefore, in the subsequent level of recursion, any item from FPTi can be appended to Pi to generate a new pattern (Aggarwal, Bhuiyan, & Hasan, 2014).

2.3. Literature on the development platform and tools.

2.3.1. PyCharm

Figure 2.6: PyCharm IDE

PyCharm is a popular Integrated Development Environment (IDE) for programming, especially the Python programming language. It is written in Java and Python and was released for the first time by JetBrains in February 2010. PyCharm is compatible with Windows, macOS, and Linux. It offers a graphical debugger, an integrated unit tester, coding aid, support for web development with Django, and integration with the data science platform provided by Anaconda (Simplilearn, 2022).

2.3.2. Mlxtend

Figure 2.7: Mlxtend

Mlxtend, stands for machine learning extensions, is a Python library of essential tools for day-to-day data science and machine learning tasks. According to (Raschka, MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack, 2018), it provides machine learning and data science utilities and extensions to Python's scientific computing stack. It was created by Sebastian Raschka in 2018.

2.3.3. NumPy

According to its official documentation, NumPy is an open-source initiative that aims to facilitate numerical computing with Python. It was founded in 2005 on the foundation of the earlier Numeric and Numarray libraries. NumPy will always be 100% open-source software, released under the permissive terms of the BSD licence with modifications.

Figure 2.8: NumPy

NumPy provides a multidimensional array object, a number of derived objects (such as masked arrays and matrices), and a number of array operations functions. These include mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, elementary linear algebra, elementary statistical operations, and random simulation, among others.

2.3.4. pandas

According to its official documentation, Pandas is a software library written for the Python programming language for data manipulation and analysis. Particularly, it provides the data structures and processes necessary for the manipulation of numerical tables and time series. It is open-source software distributed with the BSD licence that has three clauses.

Figure 2.9: pandas

Pandas is useful when performing association analysis because it provides a DataFrame object that is fast and efficient for data manipulation, complete with inbuilt indexing. Furthermore, it has tools for reading and writing data between data structures stored in memory and a variety of formats, including CSV and text files, Microsoft Excel, SQL databases, and the HDF5 format. Next, pandas can handle missing data by gaining an automatic label-based alignment in computations and simply modifying the messy data into an ordered form with intelligent data alignment. Also, pandas is appropriate when data sets are capable of undergoing flexible reshaping and pivoting.

2.3.5. Matplotlib

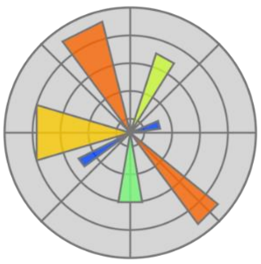


Figure 2.10: Matplotlib

According to its official documentation, Matplotlib is a plotting library for the Python computer language and its NumPy numerical mathematics extension. It provides an object-oriented API for integrating plots into programmes that use GUI toolkits such as Tkinter, wxPython, Qt, or GTK. Matplotlib is an exhaustive Python toolkit for building static, animated, and interactive visualisations. In this context, Matplotlib can provide intuitive heatmaps and bar charts to visualise the association rules and frequency of items respectively.

2.3.6. seaborn

Figure 2.11: Seaborn

According to its official documentation, Seaborn is a Python library for plotting data. It is based on the Matplotlib library. It gives programmers a high-level interface for making statistical graphics that look good and intuitive. Programmers may quickly plot their data when they use the seaborn library because it contains the necessary tools. As a result of the fact that this library is used to visualise the, programmers are relieved of the responsibility of attending to the internal intricacies. All that is required of them is to provide our data set or data into the relplot() method, and it will automatically calculate and place the value in the appropriate location.

Chapter 3: Methodology

3.1. Structure and organization of the dataset with description

3.1.1. Background

Figure 3.1: The Bread Basket

The dataset is obtained from a bakery "The Bread Basket" in Edinburg, UK. Figure above shows the bakery obtained from Facebook. The dataset is obtained from Kaggle under CC0: Public Domain license. The dataset provides transaction information for clients who made purchases from this bakery between 30th October 2016 and 4th September 2017. The dataset is a csv file that consists of over 9000 transactions, 20507 records, and 4 columns make up the dataset. The header consists of the following columns:

1. TransactionNo : A unique identifier for each purchase transaction
2. Items : Name of item purchased
3. DateTime : Date and time of which the transaction was made
4. Daypart : Time of day (morning, afternoon, evening, night)
5. DayType : Day of week (weekend or weekdays)

3.1.2. Data Preprocessing

```
TransactionNo,Items,DateTime,Daypart,DayType
1,Bread,2016-10-30 09:58:11,Morning,Weekend
2,Scandinavian,2016-10-30 10:05:34,Morning,Weekend
2,Scandinavian,2016-10-30 10:05:34,Morning,Weekend
3,Hot chocolate,2016-10-30 10:07:57,Morning,Weekend
3,Jam,2016-10-30 10:07:57,Morning,Weekend
3,Cookies,2016-10-30 10:07:57,Morning,Weekend
4,Muffin,2016-10-30 10:08:41,Morning,Weekend
5,Coffee,2016-10-30 10:13:03,Morning,Weekend
5,Pastry,2016-10-30 10:13:03,Morning,Weekend
5,Bread,2016-10-30 10:13:03,Morning,Weekend
6,Medialuna,2016-10-30 10:16:55,Morning,Weekend
6,Pastry,2016-10-30 10:16:55,Morning,Weekend
6,Muffin,2016-10-30 10:16:55,Morning,Weekend
…
```

The above lines are extracted from the dataset. A purchase transaction may contain one or more items. Every row represents an item purchased through a transaction. Row 1 with TransactionNo 1 consists of only one item, which is Bread. Row 2 and 3 are from another transaction with TransactionNo 2. Simply put, rows with the same TransactionNo are of the same transaction.

The aim of this data prepossessing is to restructure the records by grouping all items bought together in a transaction. To do that, the codes to restructure the data are written in preprocessing.py. The codes consist of reading the original .csv file and loading the data into memory. Then, an empty list is created. After that, an iterator loops through the data to group all items to the same row with same TransactionNo. Then, the populated list is written into a new .csv file. After running preprocessing.py, the extracted lines now become:

```
Bread
Scandinavian,Scandinavian
Hot chocolate,Jam,Cookies
Muffin
Coffee,Pastry,Bread
Medialuna,Pastry,Muffin
…
```

3.2. Design and development of the Apriori module

The Apriori module is derived from the Mlxtend library. According to the official documentation, its author is Sebastian Raschka and licensed under BSD 3 clause. The module provides an Apriori function to extract frequent itemsets for association rule mining. The module can be called through a function called `apriori` that takes the following parameters:

1. `df` : A pandas DataFrame in encoded format. The allowed values are either 0/1 or True/False.
2. `min_support` : A float between 0 and 1 for minimum support of the frequent itemsets returned. The default value is 0.5.
3. `use_colnames` : A boolean. If True, the function uses the dataframe's column names in the returned DataFrame instead of column indices. The default value is False.
4. `max_len` : An integer that sets the maximum length of the itemsets generated. If set None by default, then all possible item sets lengths are examined.
5. `verbose`: An integer that shows the number of iterations if it is more than or equal to 1 and low_memory is set to True. If it is 1 and low_memory is False, then it shows the number of combinations.
6. `low_memory` : A boolean. If True, it uses an iterator to search for combinations above min_support. Because this approach is roughly 3-6 times slower than the default, low memory=True should only be used for large datasets if memory resources are limited.

The function returns a pandas DataFrame with columns ['support', 'itemsets'] for all itemsets that are >= `min_support` and < than `max_len` (if `max_len` is not None). Each itemset in the itemsets column is of the built-in Python type frozenset, which functions similar to sets but is immutable.

3.3. Design and development of the Frequent Pattern Growth module

Frequent Pattern Growth module is derived from the Mlxtend library. According to the official documentation, its author is Steve Harenberg and licensed under BSD 3 clause. The module provides an FP-Growth function to extract frequent itemsets for association rule mining. In contrast to Apriori, FP-Growth does not need candidate generation. It is especially appealing for huge datasets since it internally uses a so-called FP-tree (frequent pattern tree) data structure without explicitly constructing the candidate sets. The module can be called through a function called `fpgrowth`. It takes the following parameters:

1. `df` : A pandas DataFrame in encoded format. The allowed values are either 0/1 or True/False.
2. `min_support` : A float between 0 and 1 for minimum support of the frequent itemsets returned. The default value is 0.5.
3. `use_colnames` : A boolean. If True, the function uses the dataframe's column names in the returned DataFrame instead of column indices. The default value is False.
4. `max_len` : An integer that sets the maximum length of the itemsets generated. If set None by default, then all possible item sets lengths are examined.
5. `verbose` : An integer that shows the stages of conditional tree generation.

The function returns a pandas DataFrame with columns ['support', 'itemsets'] for all itemsets that are >= `min_support` and < than `max_len` (if `max_len` is not None). Each itemset in the itemsets column is of the built-in Python type frozenset, which functions similar to sets but is immutable.

Chapter 4: Discussion and Analysis

4.1. Prerequisites

The following are the Python libraries used in the program as specified in `requirements.txt`:

1. numpy
2. pandas
3. matplotlib
4. mlxtend
5. seaborn
6. warn

4.2. Data Extraction

4.2.1. Terminal Keyboard

In the program, there are two options for users to input their records. The first is by using **terminal keyboard** whereby users type in the records into terminal or command line prompt. Users can either manually type all the item columns and transaction rows, or copy and paste the records. Users must separate each item using comma (‘,’) and each translation using newline (ENTER) Every line of transactions given by the users is stored in a list. Users should type in exactly ‘-1’ on a newline and immediately followed by ENTER to stop the program from receiving records.

Then, the program generates a 5-character string using ASCII letters. It uses this string to create a new .csv file and rename it. Once a file is created, the program writes every line from the list into the file. Every item within a line is converted to uppercase using list comprehensions. Once everything is done, a new .csv file will be stored within the directory and ready for further processing.

4.2.2. Input File

The second method is **input file**. The program prompts users to specify a valid path, either relative or absolute path. The file must end with .csv filetype. Once the path is validated, the .csv file will be loaded in memory and ready for further processing.

4.3. Data Transformation

Since the csv file would have a different number of columns for each row, therefore the first row determines the number of items per row. This creates an error tokenising data when succeeding rows have more items than the first row. A simple trick would be to supply dynamically generated column names. The following code snippet creates columns with length of the row with highest item count.

```
largest_column_count = 0
with open(csv, 'r') as temp_f:
    lines = temp_f.readlines()
    for l in lines:
        # Count the column count for the current line
        column_count = len(l.split(',')) + 1
        # Set the new most column count
        largest_column_count = column_count if largest_column_count < column_count
else largest_column_count

# Generate column names (will be 0, 1, 2, ..., largest_column_count - 1)
column_names = [i for i in range(0, largest_column_count)]
```

After that, the column_names is passed as an argument into a function to read the csv file.

```
dataset = pd.read_csv(csv, header=None, delimiter=',', names=column_names)
```

Before using any association rule mining algorithms, the data needs to be transformed into data frame format. The following code snippet transforms the data from csv file into dataframe format.

```
# Transform data into dataframe

df_out = dataset.apply(lambda x: list(x.dropna().values), axis=1).to_list()
te = TransactionEncoder()
te_ary = te.fit(df_out).transform(df_out)
dataframe = pd.DataFrame(te_ary, columns=te.columns_)
```

4.4. Data Analysis

Now that the dataframe is loaded in memory, it is time to perform association analysis and simple data visualisation techniques.

Users are given three options:

1. Perform association analysis
2. View the frequency of n-th most purchased items using bar chart
3. View the frequency of n-th least purchased items using bar chart

4.4.1. Simple Data Visualisation

Using bar charts, users can know at a glance the items with the highest frequency or lowest frequency. Users can specify the n value. The n value is meant to determine the number of items displayed as bar columns. For example, if users specify n as 10, then users can see the Top 10 most frequent items, or Bottom 10 least frequent items. The constraints of n are that it cannot be zero or lower, and exceed the number of items in the datasets.

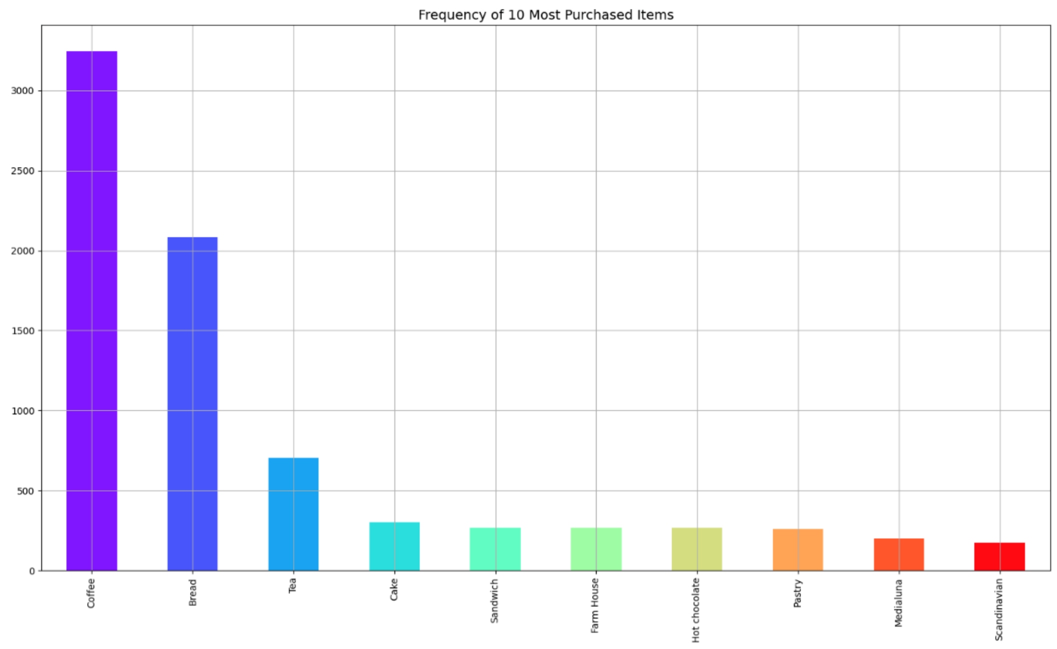


Figure 4.1: Top 10 Products

Using Bread Basket dataset, the top 10 most purchased items are shown in Figure above. The top 3 items are coffee, bread, and tea. Association analysis can uncover other products which are often purchased together with these three most popular products. They serve as a good starting point.

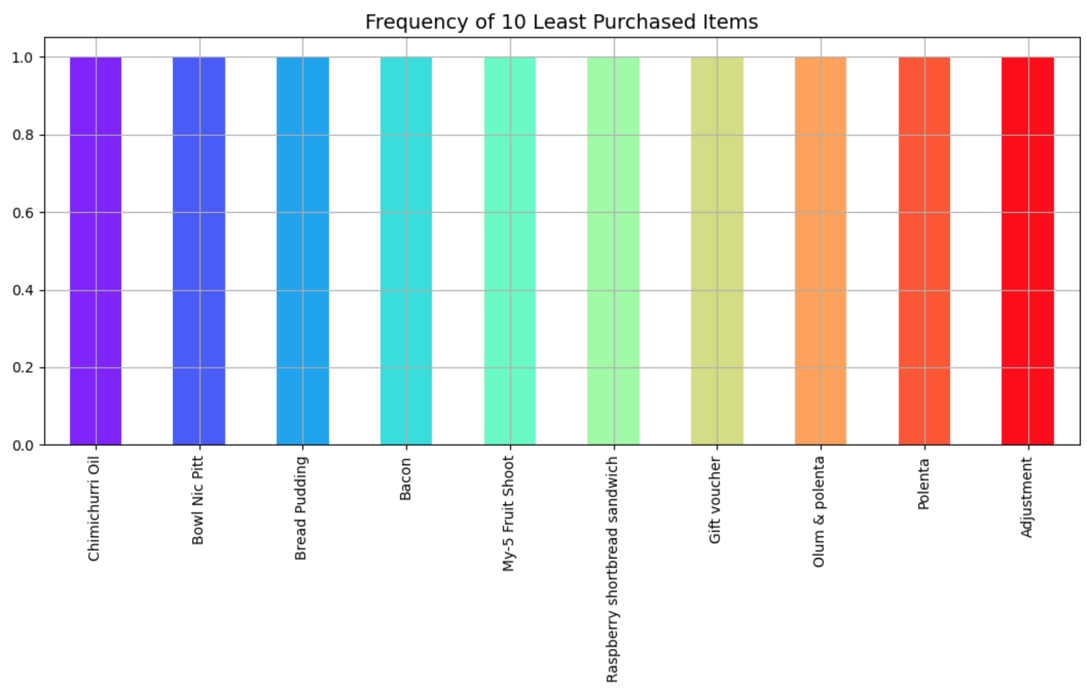


Figure 4.2: Bottom 10 Products

Figure above shows the 10 least purchased items. Each of them was purchased only one in the 9683 transactions, thus they can be ignored later on during association analysis.

Though these bar charts offer a quick, interesting view of the best and least performing items, they do not offer insights into ways to leverage the bestselling items to improve the sales. However, they provide good starting points on the products or items to focus since association analysis typically deals with hundreds, or thousands, of records at once. This is when association analysis has to be performed.

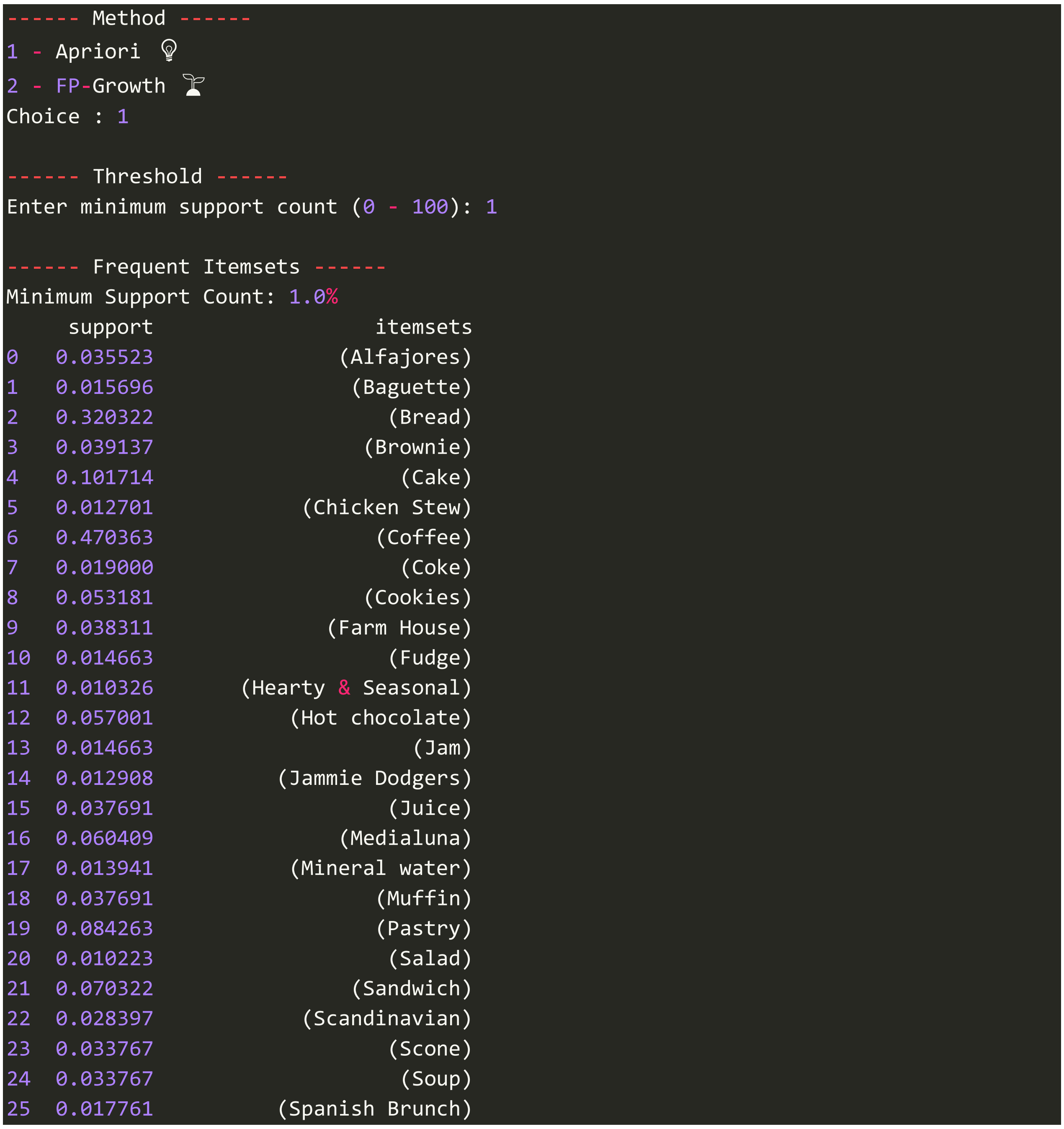
4.4.2. Association Analysis

#1: Generate Frequent Itemsets

Users are given two options to obtain frequent itemsets from the records: Apriori and FP growth. Then, users are prompted to enter the minimum support count (in percent %). This value is a floating-point value from 0.0 to 100.0. 1% is a good starting point for the first association analysis. 1% out of 9684 transactions yields roughly 96 transactions.

Thus, using 1% minimum support, the following outputs are the frequent itemsets generated using Apriori and FP growth respectively:

Apriori



```
------ Method ------
1 - Apriori 💡
2 - FP-Growth 🌱
Choice : 1

------ Threshold ------
Enter minimum support count (0 - 100): 1

------ Frequent Itemsets ------
Minimum Support Count: 1.0%
     support                 itemsets
0   0.035523              (Alfajores)
1   0.015696               (Baguette)
2   0.320322                  (Bread)
3   0.039137                (Brownie)
4   0.101714                   (Cake)
5   0.012701           (Chicken Stew)
6   0.470363                 (Coffee)
7   0.019000                   (Coke)
8   0.053181                (Cookies)
9   0.038311             (Farm House)
10  0.014663                  (Fudge)
11  0.010326      (Hearty & Seasonal)
12  0.057001          (Hot chocolate)
13  0.014663                    (Jam)
14  0.012908         (Jammie Dodgers)
15  0.037691                  (Juice)
16  0.060409              (Medialuna)
17  0.013941          (Mineral water)
18  0.037691                 (Muffin)
19  0.084263                 (Pastry)
20  0.010223                  (Salad)
21  0.070322               (Sandwich)
22  0.028397           (Scandinavian)
23  0.033767                  (Scone)
24  0.033767                   (Soup)
25  0.017761         (Spanish Brunch)
```

```
26  0.139715                     (Tea)
27  0.015076                  (Tiffin)
28  0.032838                   (Toast)
29  0.019827                (Truffles)
30  0.019104       (Coffee, Alfajores)
31  0.010223          (Bread, Brownie)
32  0.022718             (Bread, Cake)
33  0.085502           (Bread, Coffee)
34  0.013941          (Bread, Cookies)
35  0.012908    (Bread, Hot chocolate)
36  0.016316        (Bread, Medialuna)
37  0.027571           (Bread, Pastry)
38  0.016006         (Bread, Sandwich)
39  0.026952              (Bread, Tea)
40  0.018897         (Brownie, Coffee)
41  0.052458            (Coffee, Cake)
42  0.010946     (Cake, Hot chocolate)
43  0.022615               (Tea, Cake)
44  0.027365         (Cookies, Coffee)
45  0.028294   (Coffee, Hot chocolate)
46  0.020033           (Coffee, Juice)
47  0.033664       (Coffee, Medialuna)
48  0.017968          (Coffee, Muffin)
49  0.045333          (Pastry, Coffee)
50  0.036762        (Coffee, Sandwich)
51  0.017451           (Scone, Coffee)
52  0.015283            (Coffee, Soup)
53  0.010533  (Spanish Brunch, Coffee)
54  0.047501             (Coffee, Tea)
55  0.022718           (Toast, Coffee)
56  0.013631           (Tea, Sandwich)
57  0.010223   (Bread, Pastry, Coffee)
```

<u>FP Growth</u>

```
------ Method ------
1 - Apriori 💡
2 - FP-Growth 🌱
Choice : 2

------ Threshold ------
Enter minimum support count (0 - 100): 1

------ Frequent Itemsets ------
Minimum Support Count: 1.0%
     support                  itemsets
0   0.320322                   (Bread)
1   0.028397            (Scandinavian)
2   0.057001           (Hot chocolate)
3   0.053181                 (Cookies)
4   0.014663                     (Jam)
```

```
5   0.037691                   (Muffin)
6   0.470363                   (Coffee)
7   0.084263                   (Pastry)
8   0.060409                (Medialuna)
9   0.139715                      (Tea)
10  0.013941            (Mineral water)
11  0.038311               (Farm House)
12  0.014663                    (Fudge)
13  0.037691                    (Juice)
14  0.010326       (Hearty & Seasonal)
15  0.033767                     (Soup)
16  0.101714                     (Cake)
17  0.019000                     (Coke)
18  0.070322                 (Sandwich)
19  0.035523                (Alfajores)
20  0.039137                  (Brownie)
21  0.019827                 (Truffles)
22  0.012908           (Jammie Dodgers)
23  0.015076                   (Tiffin)
24  0.032838                    (Toast)
25  0.033767                    (Scone)
26  0.010223                    (Salad)
27  0.012701             (Chicken Stew)
28  0.017761           (Spanish Brunch)
29  0.015696                 (Baguette)
30  0.085502            (Bread, Coffee)
31  0.028294   (Coffee, Hot chocolate)
32  0.012908    (Bread, Hot chocolate)
33  0.010946     (Cake, Hot chocolate)
34  0.027365          (Cookies, Coffee)
35  0.013941           (Bread, Cookies)
36  0.017968           (Coffee, Muffin)
37  0.045333           (Pastry, Coffee)
38  0.027571            (Bread, Pastry)
39  0.010223   (Bread, Pastry, Coffee)
40  0.033664        (Coffee, Medialuna)
41  0.016316         (Bread, Medialuna)
42  0.047501              (Coffee, Tea)
43  0.026952               (Bread, Tea)
44  0.020033            (Coffee, Juice)
45  0.015283             (Coffee, Soup)
46  0.052458             (Coffee, Cake)
47  0.022615                (Tea, Cake)
48  0.022718              (Bread, Cake)
49  0.013631            (Tea, Sandwich)
50  0.016006          (Bread, Sandwich)
51  0.036762         (Coffee, Sandwich)
52  0.019104        (Coffee, Alfajores)
53  0.018897          (Brownie, Coffee)
54  0.010223           (Bread, Brownie)
55  0.022718            (Toast, Coffee)
56  0.017451            (Scone, Coffee)
57  0.010533  (Spanish Brunch, Coffee)
```

Both methods yielded the same 57 frequent itemsets with the same 1% minimum support count. From here, association rules can be generated using the frequent itemsets.

#2: Generate Association Rules

The frequent itemsets are passed into `run_association_rules()` that generates rules, filters results, and generate heatmaps. Firstly, all possible association rules between every frequent itemset are generated by the following code snippet. This is done by setting the minimum threshold for support to 0, thus all possible rules can be generated.

```
rules = association_rules(frq_items, metric='support', min_threshold=0.0)
```

#3: Filter using Metrics

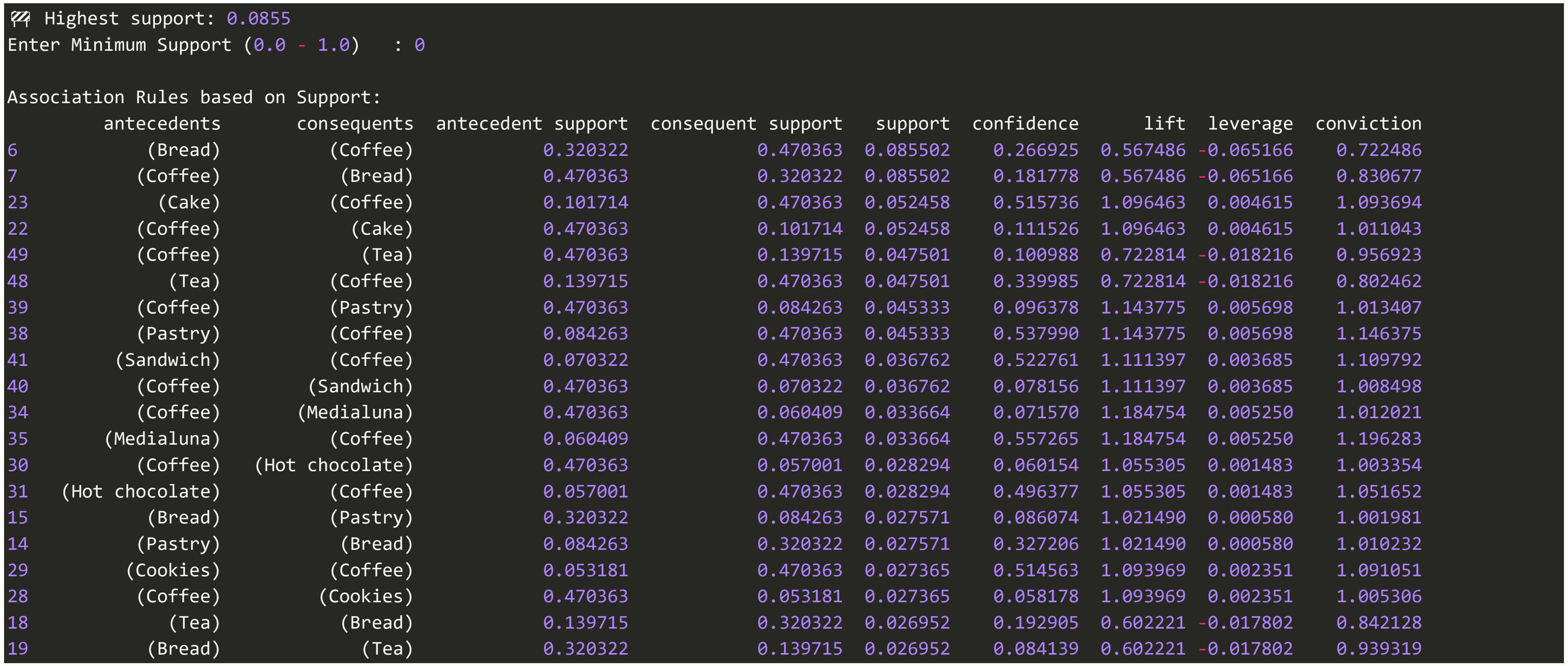
After that, users are given options to filter based on their desired threshold using either support, confidence, lift, or all three at once.

<u>Support</u>

Minimum Threshold: 0.0 or 0%

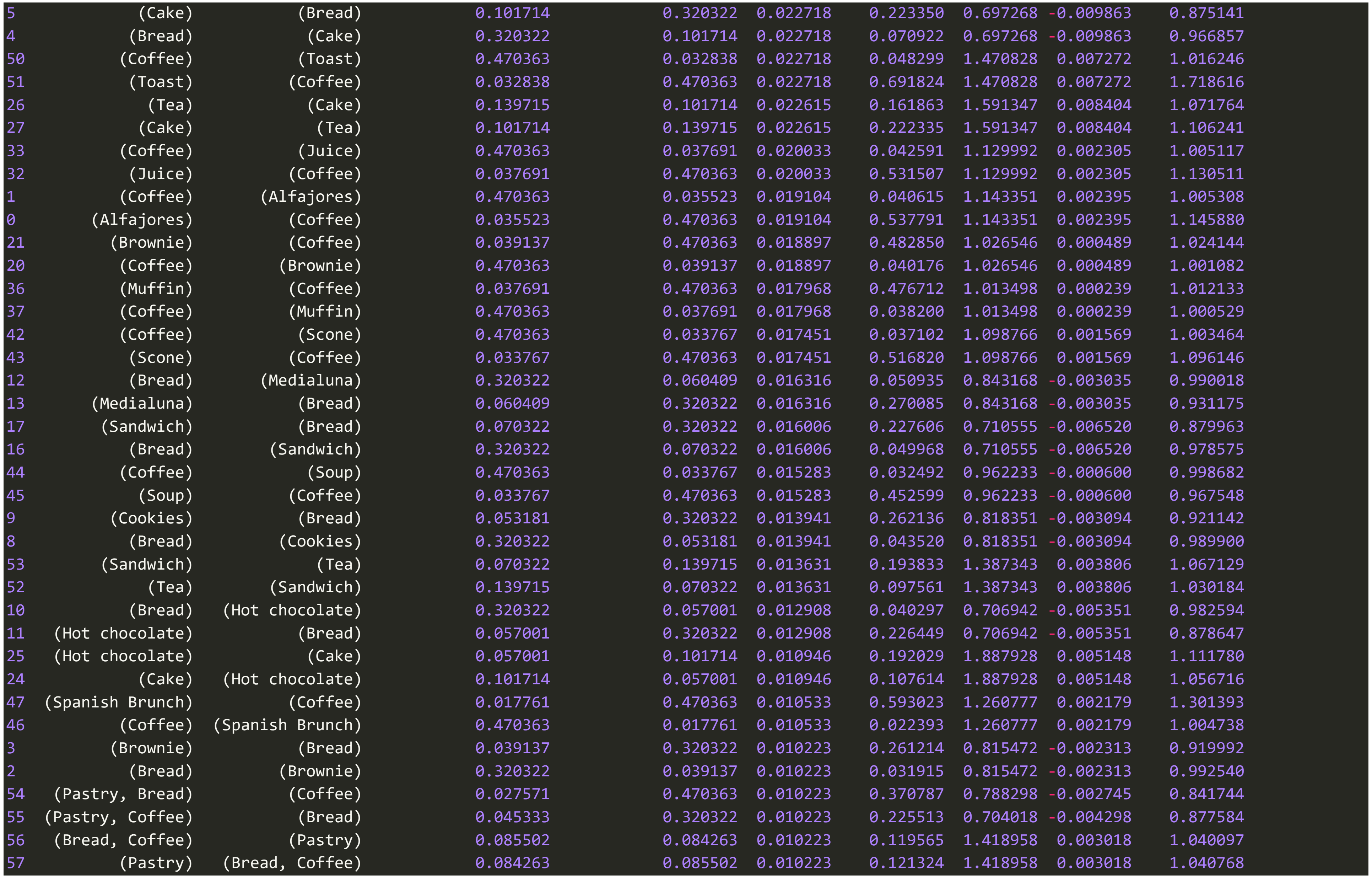
Justification: This is to generate all possible rules for the frequent itemsets. This helps to determine the filter values for confidence and lift later on.

Association Rules:



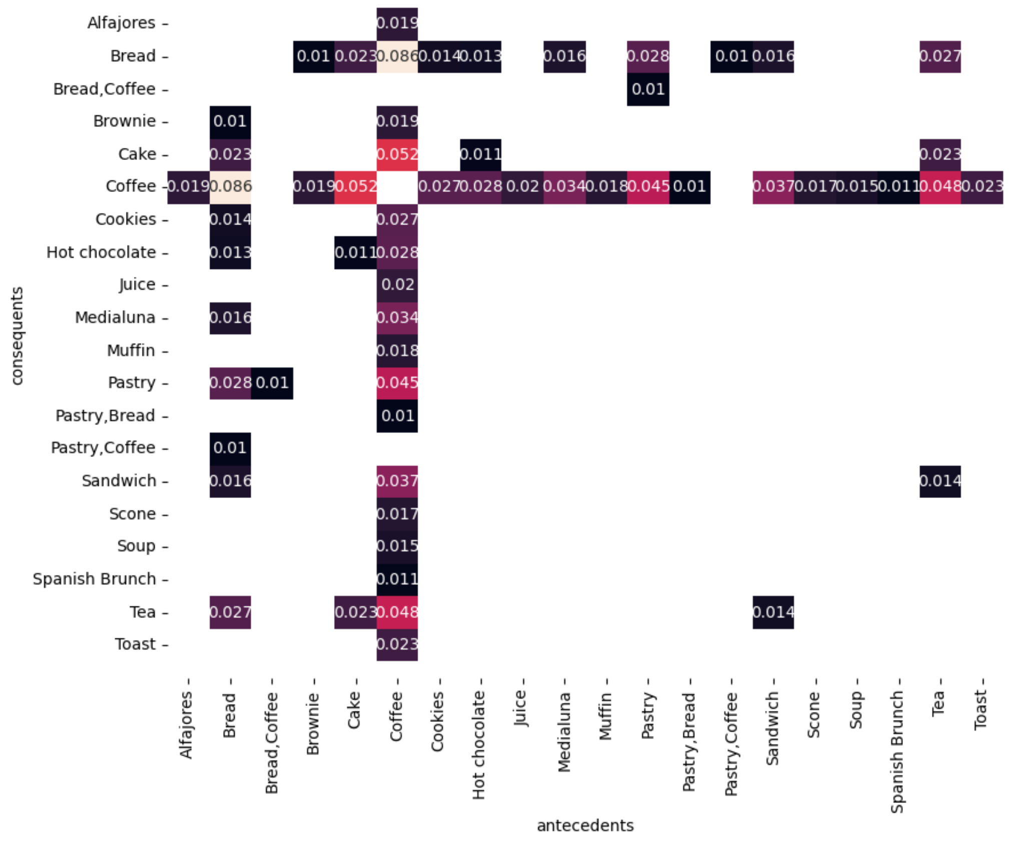
```
🚧 Highest support: 0.0855
Enter Minimum Support (0.0 - 1.0)   : 0

Association Rules based on Support:
         antecedents       consequents  antecedent support  consequent support   support  confidence      lift  leverage  conviction
6            (Bread)          (Coffee)            0.320322            0.470363  0.085502    0.266925  0.567486 -0.065166    0.722486
7           (Coffee)           (Bread)            0.470363            0.320322  0.085502    0.181778  0.567486 -0.065166    0.830677
23            (Cake)          (Coffee)            0.101714            0.470363  0.052458    0.515736  1.096463  0.004615    1.093694
22          (Coffee)            (Cake)            0.470363            0.101714  0.052458    0.111526  1.096463  0.004615    1.011043
49          (Coffee)             (Tea)            0.470363            0.139715  0.047501    0.100988  0.722814 -0.018216    0.956923
48             (Tea)          (Coffee)            0.139715            0.470363  0.047501    0.339985  0.722814 -0.018216    0.802462
39          (Coffee)          (Pastry)            0.470363            0.084263  0.045333    0.096378  1.143775  0.005698    1.013407
38          (Pastry)          (Coffee)            0.084263            0.470363  0.045333    0.537990  1.143775  0.005698    1.146375
41        (Sandwich)          (Coffee)            0.070322            0.470363  0.036762    0.522761  1.111397  0.003685    1.109792
40          (Coffee)        (Sandwich)            0.470363            0.070322  0.036762    0.078156  1.111397  0.003685    1.008498
34          (Coffee)       (Medialuna)            0.470363            0.060409  0.033664    0.071570  1.184754  0.005250    1.012021
35       (Medialuna)          (Coffee)            0.060409            0.470363  0.033664    0.557265  1.184754  0.005250    1.196283
30          (Coffee)   (Hot chocolate)            0.470363            0.057001  0.028294    0.060154  1.055305  0.001483    1.003354
31   (Hot chocolate)          (Coffee)            0.057001            0.470363  0.028294    0.496377  1.055305  0.001483    1.051652
15           (Bread)          (Pastry)            0.320322            0.084263  0.027571    0.086074  1.021490  0.000580    1.001981
14          (Pastry)           (Bread)            0.084263            0.320322  0.027571    0.327206  1.021490  0.000580    1.010232
29         (Cookies)          (Coffee)            0.053181            0.470363  0.027365    0.514563  1.093969  0.002351    1.091051
28          (Coffee)         (Cookies)            0.470363            0.053181  0.027365    0.058178  1.093969  0.002351    1.005306
18             (Tea)           (Bread)            0.139715            0.320322  0.026952    0.192905  0.602221 -0.017802    0.842128
19           (Bread)             (Tea)            0.320322            0.139715  0.026952    0.084139  0.602221 -0.017802    0.939319
```


```
5            (Cake)          (Bread)     0.101714     0.320322  0.022718   0.223350  0.697268 -0.009863    0.875141
4           (Bread)           (Cake)     0.320322     0.101714  0.022718   0.070922  0.697268 -0.009863    0.966857
50         (Coffee)          (Toast)     0.470363     0.032838  0.022718   0.048299  1.470828  0.007272    1.016246
51          (Toast)         (Coffee)     0.032838     0.470363  0.022718   0.691824  1.470828  0.007272    1.718616
26            (Tea)           (Cake)     0.139715     0.101714  0.022615   0.161863  1.591347  0.008404    1.071764
27           (Cake)            (Tea)     0.101714     0.139715  0.022615   0.222335  1.591347  0.008404    1.106241
33         (Coffee)          (Juice)     0.470363     0.037691  0.020033   0.042591  1.129992  0.002305    1.005117
32          (Juice)         (Coffee)     0.037691     0.470363  0.020033   0.531507  1.129992  0.002305    1.130511
1          (Coffee)      (Alfajores)     0.470363     0.035523  0.019104   0.040615  1.143351  0.002395    1.005308
0       (Alfajores)         (Coffee)     0.035523     0.470363  0.019104   0.537791  1.143351  0.002395    1.145880
21        (Brownie)         (Coffee)     0.039137     0.470363  0.018897   0.482850  1.026546  0.000489    1.024144
20         (Coffee)        (Brownie)     0.470363     0.039137  0.018897   0.040176  1.026546  0.000489    1.001082
36         (Muffin)         (Coffee)     0.037691     0.470363  0.017968   0.476712  1.013498  0.000239    1.012133
37         (Coffee)         (Muffin)     0.470363     0.037691  0.017968   0.038200  1.013498  0.000239    1.000529
42         (Coffee)          (Scone)     0.470363     0.033767  0.017451   0.037102  1.098766  0.001569    1.003464
43          (Scone)         (Coffee)     0.033767     0.470363  0.017451   0.516820  1.098766  0.001569    1.096146
12          (Bread)      (Medialuna)     0.320322     0.060409  0.016316   0.050935  0.843168 -0.003035    0.990018
13      (Medialuna)          (Bread)     0.060409     0.320322  0.016316   0.270085  0.843168 -0.003035    0.931175
17       (Sandwich)          (Bread)     0.070322     0.320322  0.016006   0.227606  0.710555 -0.006520    0.879963
16          (Bread)       (Sandwich)     0.320322     0.070322  0.016006   0.049968  0.710555 -0.006520    0.978575
44         (Coffee)           (Soup)     0.470363     0.033767  0.015283   0.032492  0.962233 -0.000600    0.998682
45           (Soup)         (Coffee)     0.033767     0.470363  0.015283   0.452599  0.962233 -0.000600    0.967548
9         (Cookies)          (Bread)     0.053181     0.320322  0.013941   0.262136  0.818351 -0.003094    0.921142
8           (Bread)        (Cookies)     0.320322     0.053181  0.013941   0.043520  0.818351 -0.003094    0.989900
53       (Sandwich)            (Tea)     0.070322     0.139715  0.013631   0.193833  1.387343  0.003806    1.067129
52            (Tea)       (Sandwich)     0.139715     0.070322  0.013631   0.097561  1.387343  0.003806    1.030184
10          (Bread)  (Hot chocolate)     0.320322     0.057001  0.012908   0.040297  0.706942 -0.005351    0.982594
11  (Hot chocolate)          (Bread)     0.057001     0.320322  0.012908   0.226449  0.706942 -0.005351    0.878647
25  (Hot chocolate)           (Cake)     0.057001     0.101714  0.010946   0.192029  1.887928  0.005148    1.111780
24           (Cake)  (Hot chocolate)     0.101714     0.057001  0.010946   0.107614  1.887928  0.005148    1.056716
47 (Spanish Brunch)         (Coffee)     0.017761     0.470363  0.010533   0.593023  1.260777  0.002179    1.301393
46         (Coffee) (Spanish Brunch)     0.470363     0.017761  0.010533   0.022393  1.260777  0.002179    1.004738
3         (Brownie)          (Bread)     0.039137     0.320322  0.010223   0.261214  0.815472 -0.002313    0.919992
2           (Bread)        (Brownie)     0.320322     0.039137  0.010223   0.031915  0.815472 -0.002313    0.992540
54  (Pastry, Bread)         (Coffee)     0.027571     0.470363  0.010223   0.370787  0.788298 -0.002745    0.841744
55 (Pastry, Coffee)          (Bread)     0.045333     0.320322  0.010223   0.225513  0.704018 -0.004298    0.877584
56  (Bread, Coffee)         (Pastry)     0.085502     0.084263  0.010223   0.119565  1.418958  0.003018    1.040097
57         (Pastry)  (Bread, Coffee)     0.084263     0.085502  0.010223   0.121324  1.418958  0.003018    1.040768
```

```
58            (Bread)  (Pastry, Coffee)             0.320322             0.045333  0.010223    0.031915  0.704018 -0.004298     0.986140
59           (Coffee)   (Pastry, Bread)             0.470363             0.027571  0.010223    0.021734  0.788298 -0.002745     0.994033
```

Heatmap



Findings

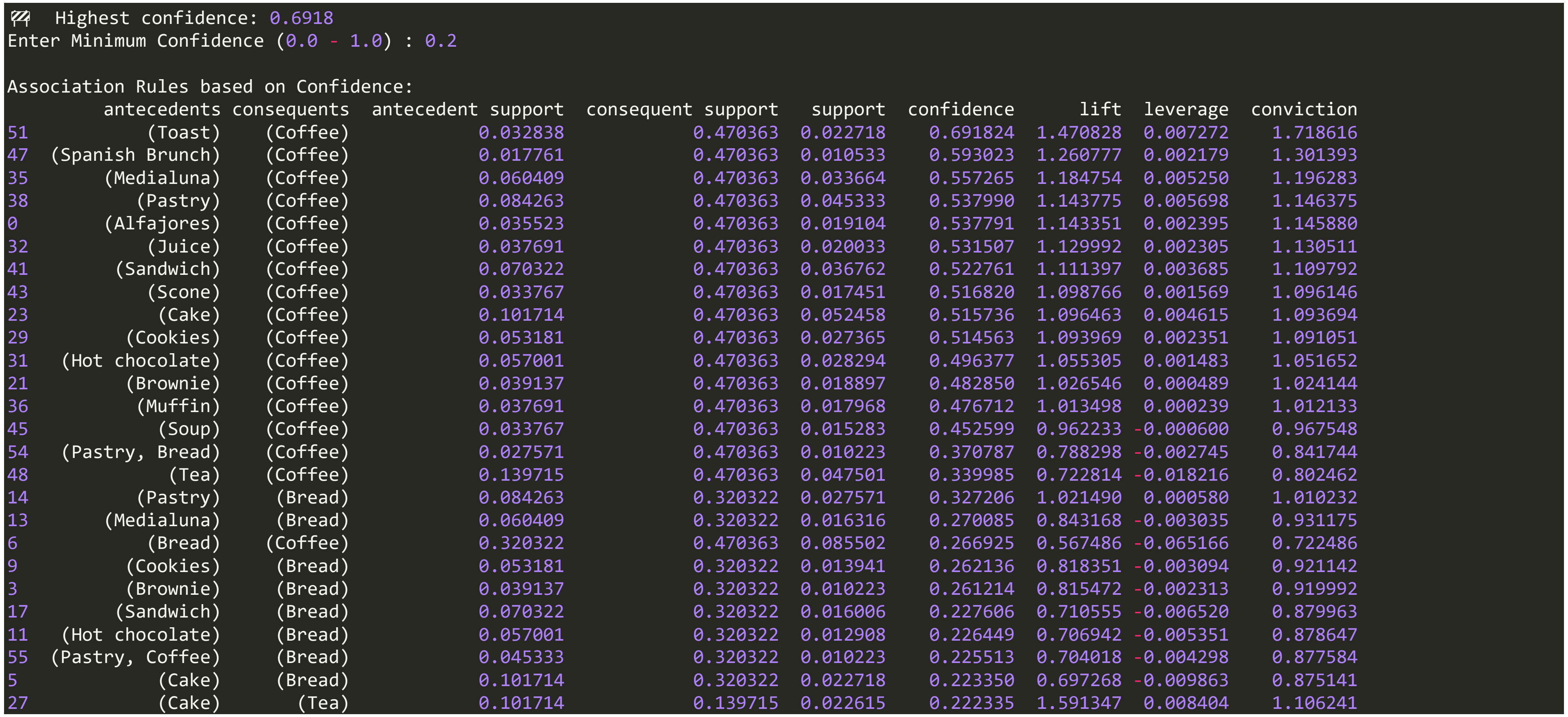
The heatmap conforms with the findings earlier for the Top 3 best-selling items, bread, coffee, and tea. These three items, especially bread and coffee, appear with other items. Also, it is noted that all three items have at least 20% the confidence level.

Confidence

Minimum Threshold: 0.2 or 20%

Justification: 20% Confidence covers all the Top 3 best selling products.

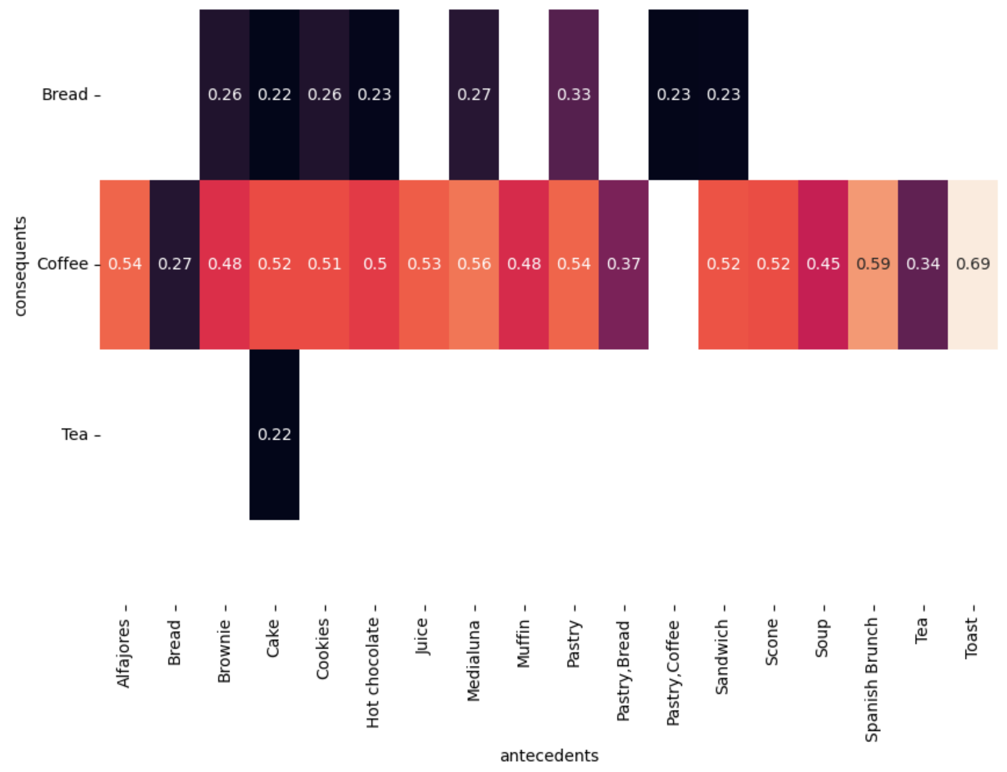
Association Rules:



```
🚧  Highest confidence: 0.6918
Enter Minimum Confidence (0.0 - 1.0) : 0.2

Association Rules based on Confidence:
         antecedents consequents  antecedent support  consequent support   support  confidence      lift  leverage  conviction
51           (Toast)    (Coffee)            0.032838            0.470363  0.022718    0.691824  1.470828  0.007272    1.718616
47   (Spanish Brunch)   (Coffee)            0.017761            0.470363  0.010533    0.593023  1.260777  0.002179    1.301393
35       (Medialuna)    (Coffee)            0.060409            0.470363  0.033664    0.557265  1.184754  0.005250    1.196283
38          (Pastry)    (Coffee)            0.084263            0.470363  0.045333    0.537990  1.143775  0.005698    1.146375
0        (Alfajores)    (Coffee)            0.035523            0.470363  0.019104    0.537791  1.143351  0.002395    1.145880
32           (Juice)    (Coffee)            0.037691            0.470363  0.020033    0.531507  1.129992  0.002305    1.130511
41        (Sandwich)    (Coffee)            0.070322            0.470363  0.036762    0.522761  1.111397  0.003685    1.109792
43           (Scone)    (Coffee)            0.033767            0.470363  0.017451    0.516820  1.098766  0.001569    1.096146
23            (Cake)    (Coffee)            0.101714            0.470363  0.052458    0.515736  1.096463  0.004615    1.093694
29         (Cookies)    (Coffee)            0.053181            0.470363  0.027365    0.514563  1.093969  0.002351    1.091051
31   (Hot chocolate)    (Coffee)            0.057001            0.470363  0.028294    0.496377  1.055305  0.001483    1.051652
21         (Brownie)    (Coffee)            0.039137            0.470363  0.018897    0.482850  1.026546  0.000489    1.024144
36          (Muffin)    (Coffee)            0.037691            0.470363  0.017968    0.476712  1.013498  0.000239    1.012133
45            (Soup)    (Coffee)            0.033767            0.470363  0.015283    0.452599  0.962233 -0.000600    0.967548
54   (Pastry, Bread)    (Coffee)            0.027571            0.470363  0.010223    0.370787  0.788298 -0.002745    0.841744
48             (Tea)    (Coffee)            0.139715            0.470363  0.047501    0.339985  0.722814 -0.018216    0.802462
14          (Pastry)     (Bread)            0.084263            0.320322  0.027571    0.327206  1.021490  0.000580    1.010232
13       (Medialuna)     (Bread)            0.060409            0.320322  0.016316    0.270085  0.843168 -0.003035    0.931175
6            (Bread)    (Coffee)            0.320322            0.470363  0.085502    0.266925  0.567486 -0.065166    0.722486
9          (Cookies)     (Bread)            0.053181            0.320322  0.013941    0.262136  0.818351 -0.003094    0.921142
3          (Brownie)     (Bread)            0.039137            0.320322  0.010223    0.261214  0.815472 -0.002313    0.919992
17        (Sandwich)     (Bread)            0.070322            0.320322  0.016006    0.227606  0.710555 -0.006520    0.879963
11   (Hot chocolate)     (Bread)            0.057001            0.320322  0.012908    0.226449  0.706942 -0.005351    0.878647
55  (Pastry, Coffee)     (Bread)            0.045333            0.320322  0.010223    0.225513  0.704018 -0.004298    0.877584
5             (Cake)     (Bread)            0.101714            0.320322  0.022718    0.223350  0.697268 -0.009863    0.875141
27            (Cake)       (Tea)            0.101714            0.139715  0.022615    0.222335  1.591347  0.008404    1.106241
```

Heatmap



Findings

- For Coffee, the following items are often bought together: Alfajores, Bread, Brownie, Cake, Hot Chocolate, Juice, Medialuna, Muffin, Pastry, Pastry & Bread, Sandwich, Scone, Soup, Spanish Brunch, Tea and Toast.
- For Bread, it is always bought with Brownie, Cake, Cookies, Hot Chocolate, Medialuna, Pastry, Pastry & Cofee, and Sandwich.
- For Tea, it is always bought with Cake.

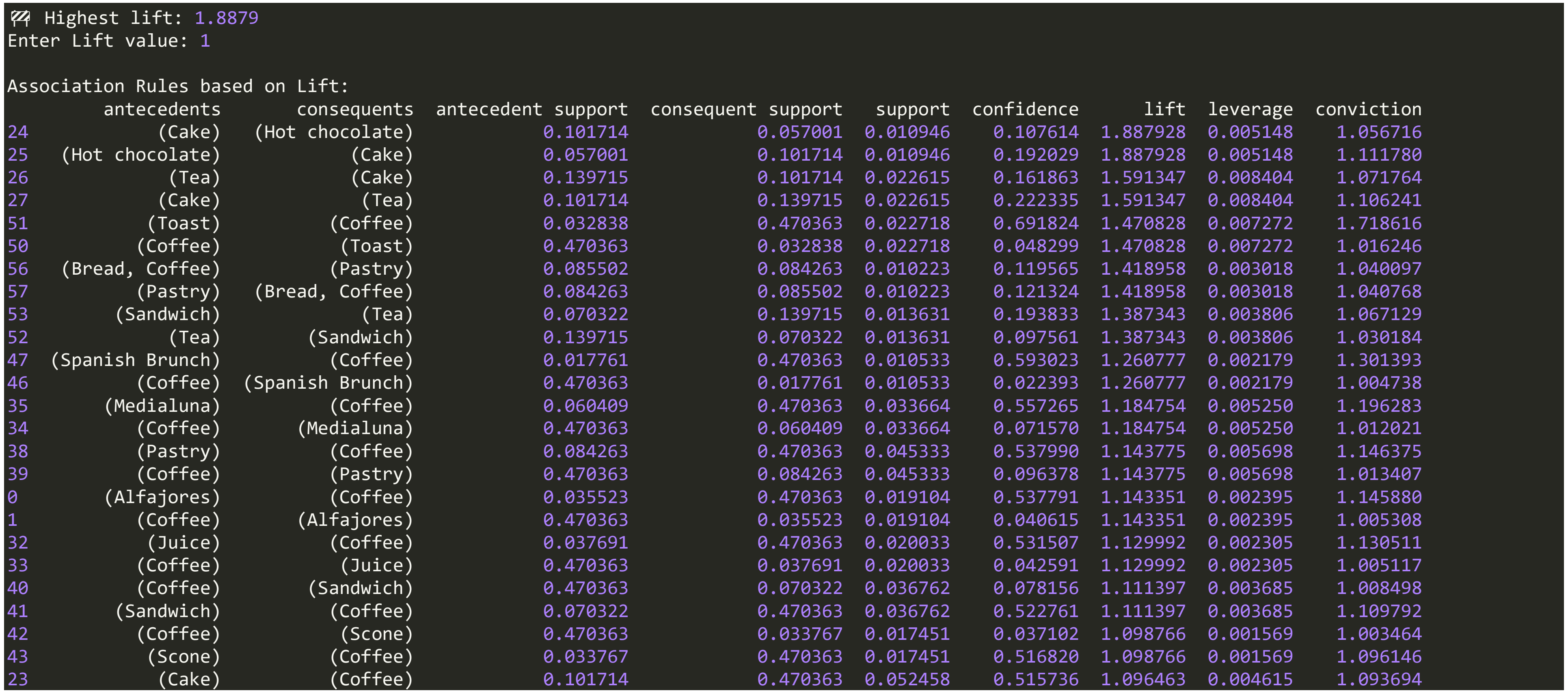
However, since the heatmap focuses on the Top 3 items, hence other possible items may be overlooked. This is when lift comes in.

Lift

Minimum Threshold: 1.0

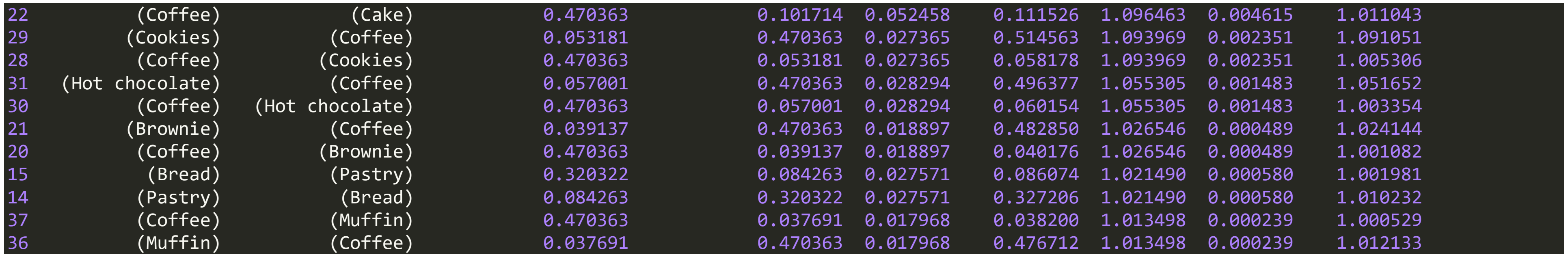
Justification: The larger the lift the greater the link between the two products. A lift value greater than 1 indicates that the rule body and the rule head appear more often together than expected, this means that the occurrence of the rule body has a positive effect on the occurrence of the rule head.

Association Rules:



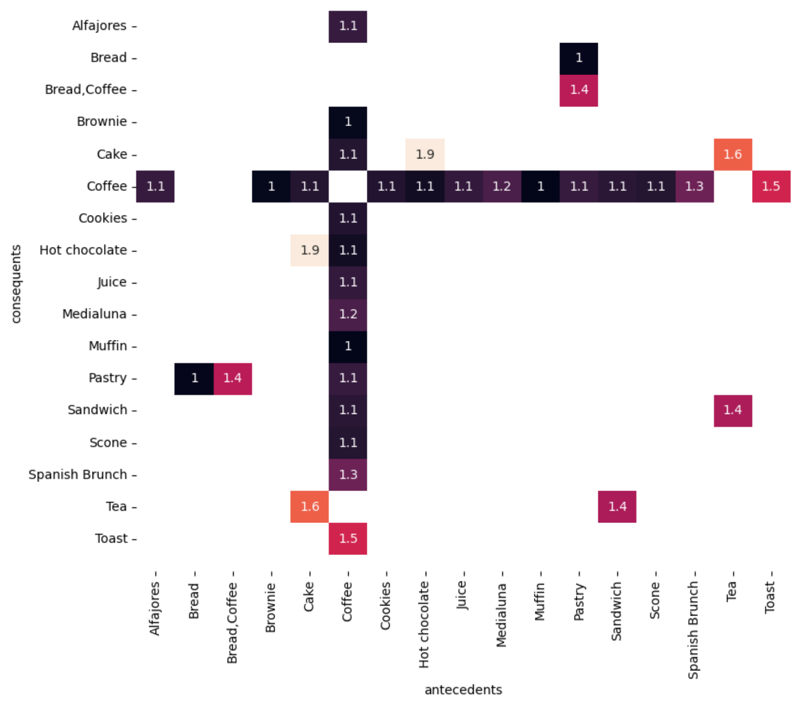
```
▨ Highest lift: 1.8879
Enter Lift value: 1

Association Rules based on Lift:
          antecedents       consequents  antecedent support  consequent support   support  confidence      lift  leverage   conviction
24             (Cake)   (Hot chocolate)            0.101714            0.057001  0.010946    0.107614  1.887928  0.005148     1.056716
25    (Hot chocolate)            (Cake)            0.057001            0.101714  0.010946    0.192029  1.887928  0.005148     1.111780
26              (Tea)            (Cake)            0.139715            0.101714  0.022615    0.161863  1.591347  0.008404     1.071764
27             (Cake)             (Tea)            0.101714            0.139715  0.022615    0.222335  1.591347  0.008404     1.106241
51            (Toast)          (Coffee)            0.032838            0.470363  0.022718    0.691824  1.470828  0.007272     1.718616
50           (Coffee)           (Toast)            0.470363            0.032838  0.022718    0.048299  1.470828  0.007272     1.016246
56    (Bread, Coffee)          (Pastry)            0.085502            0.084263  0.010223    0.119565  1.418958  0.003018     1.040097
57           (Pastry)   (Bread, Coffee)            0.084263            0.085502  0.010223    0.121324  1.418958  0.003018     1.040768
53         (Sandwich)             (Tea)            0.070322            0.139715  0.013631    0.193833  1.387343  0.003806     1.067129
52              (Tea)        (Sandwich)            0.139715            0.070322  0.013631    0.097561  1.387343  0.003806     1.030184
47   (Spanish Brunch)          (Coffee)            0.017761            0.470363  0.010533    0.593023  1.260777  0.002179     1.301393
46           (Coffee)  (Spanish Brunch)            0.470363            0.017761  0.010533    0.022393  1.260777  0.002179     1.004738
35        (Medialuna)          (Coffee)            0.060409            0.470363  0.033664    0.557265  1.184754  0.005250     1.196283
34           (Coffee)       (Medialuna)            0.470363            0.060409  0.033664    0.071570  1.184754  0.005250     1.012021
38           (Pastry)          (Coffee)            0.084263            0.470363  0.045333    0.537990  1.143775  0.005698     1.146375
39           (Coffee)          (Pastry)            0.470363            0.084263  0.045333    0.096378  1.143775  0.005698     1.013407
0         (Alfajores)          (Coffee)            0.035523            0.470363  0.019104    0.537791  1.143351  0.002395     1.145880
1            (Coffee)       (Alfajores)            0.470363            0.035523  0.019104    0.040615  1.143351  0.002395     1.005308
32            (Juice)          (Coffee)            0.037691            0.470363  0.020033    0.531507  1.129992  0.002305     1.130511
33           (Coffee)           (Juice)            0.470363            0.037691  0.020033    0.042591  1.129992  0.002305     1.005117
40           (Coffee)        (Sandwich)            0.470363            0.070322  0.036762    0.078156  1.111397  0.003685     1.008498
41         (Sandwich)          (Coffee)            0.070322            0.470363  0.036762    0.522761  1.111397  0.003685     1.109792
42           (Coffee)           (Scone)            0.470363            0.033767  0.017451    0.037102  1.098766  0.001569     1.003464
43            (Scone)          (Coffee)            0.033767            0.470363  0.017451    0.516820  1.098766  0.001569     1.096146
23             (Cake)          (Coffee)            0.101714            0.470363  0.052458    0.515736  1.096463  0.004615     1.093694
```



```
22          (Coffee)          (Cake)        0.470363        0.101714  0.052458   0.111526  1.096463  0.004615   1.011043
29         (Cookies)        (Coffee)        0.053181        0.470363  0.027365   0.514563  1.093969  0.002351   1.091051
28          (Coffee)       (Cookies)        0.470363        0.053181  0.027365   0.058178  1.093969  0.002351   1.005306
31    (Hot chocolate)       (Coffee)        0.057001        0.470363  0.028294   0.496377  1.055305  0.001483   1.051652
30          (Coffee)  (Hot chocolate)       0.470363        0.057001  0.028294   0.060154  1.055305  0.001483   1.003354
21         (Brownie)        (Coffee)        0.039137        0.470363  0.018897   0.482850  1.026546  0.000489   1.024144
20          (Coffee)       (Brownie)        0.470363        0.039137  0.018897   0.040176  1.026546  0.000489   1.001082
15           (Bread)        (Pastry)        0.320322        0.084263  0.027571   0.086074  1.021490  0.000580   1.001981
14          (Pastry)         (Bread)        0.084263        0.320322  0.027571   0.327206  1.021490  0.000580   1.010232
37          (Coffee)        (Muffin)        0.470363        0.037691  0.017968   0.038200  1.013498  0.000239   1.000529
36          (Muffin)        (Coffee)        0.037691        0.470363  0.017968   0.476712  1.013498  0.000239   1.012133
```

Heatmap



Findings

- Cake - Hot Chocolate pairs have the highest lift value
- For Coffee, the following items are bought more often together than expected: Toast, Spanish Brunch, Medialuna, Pastry, Alfajores, Juice, Sandwich, Scone, Cake, Cookies, Hot chocolate, Brownie, and Muffin
- For Bread, it is always bought with Pastry.
- For Tea, it is always bought with Cake and Sandwich.

Thus, the pattern here is that customers typically purchase something to eat with something to drink. Coffee is a more popular choice for drink to complement with food. Cake is always bought together with a drink.

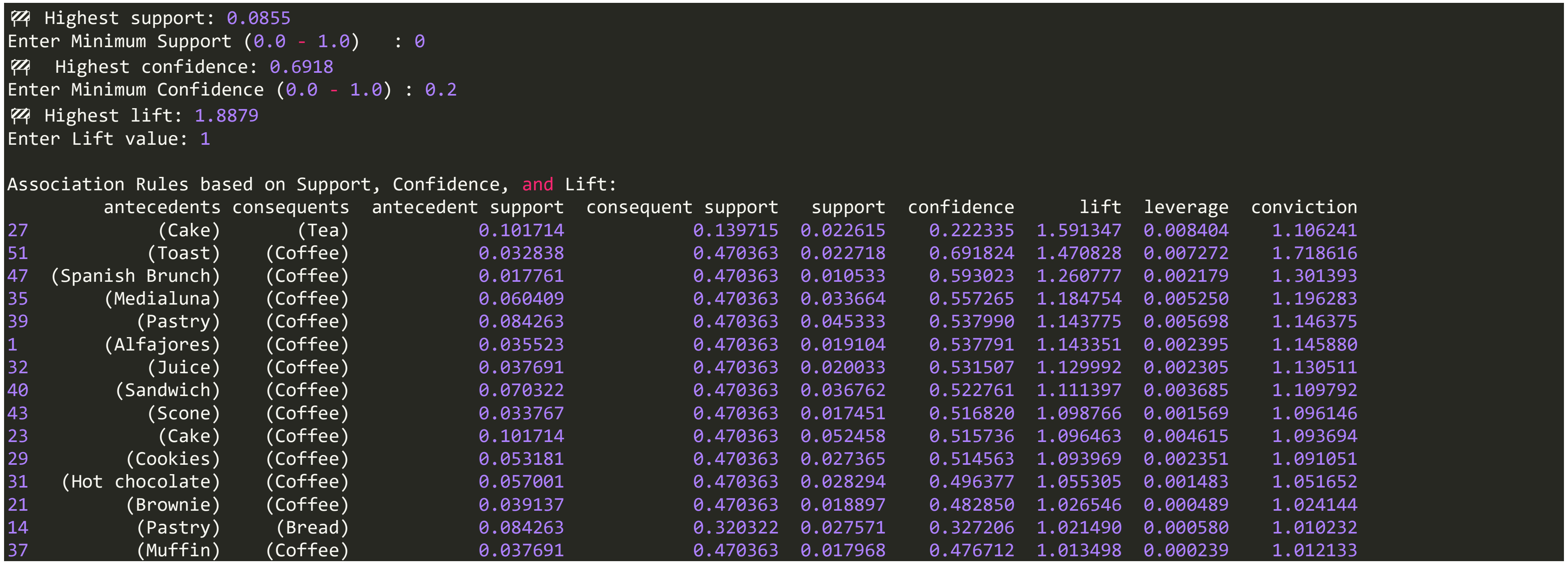
Support, Confidence, and Lift

Minimum Threshold:

- Support: 0.0 or 0%
- Confidence: 0.2 or 20%
- Lift: 1.0

Justification: Combining confidence and lift would give a better insight than either confidence or lift alone. This is basically the combination of the previous two analysis.

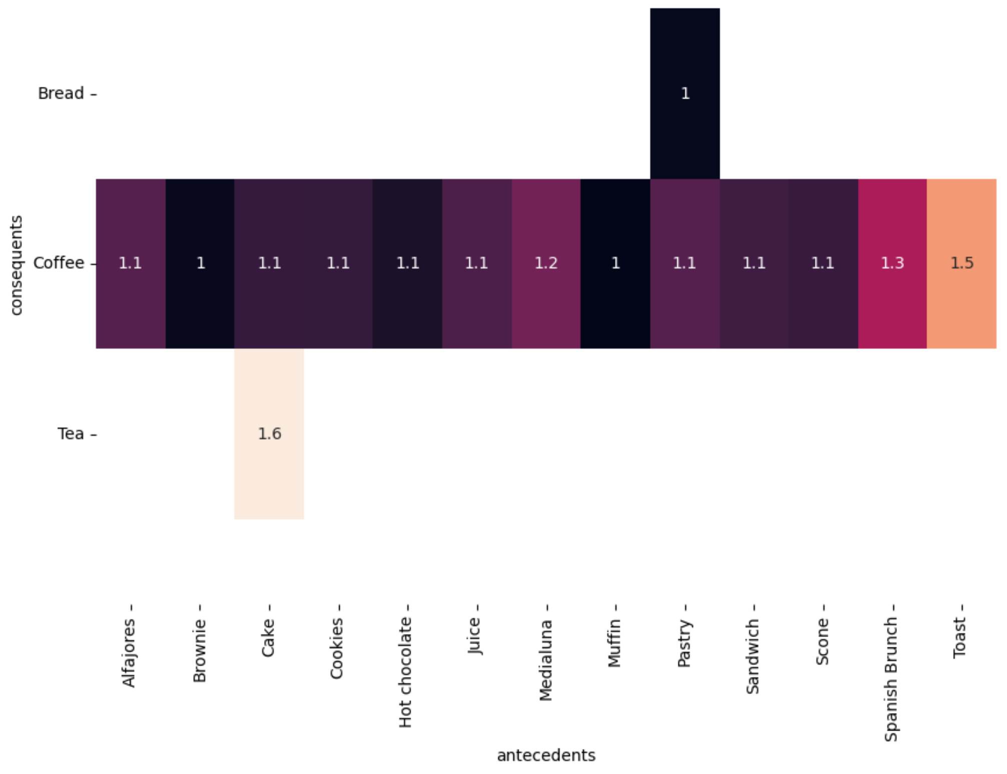
Association Rules:



```
🚧 Highest support: 0.0855
Enter Minimum Support (0.0 - 1.0)   : 0
🚧  Highest confidence: 0.6918
Enter Minimum Confidence (0.0 - 1.0) : 0.2
🚧 Highest lift: 1.8879
Enter Lift value: 1

Association Rules based on Support, Confidence, and Lift:
         antecedents consequents  antecedent support  consequent support   support  confidence      lift  leverage  conviction
27            (Cake)       (Tea)            0.101714            0.139715  0.022615    0.222335  1.591347  0.008404    1.106241
51           (Toast)    (Coffee)            0.032838            0.470363  0.022718    0.691824  1.470828  0.007272    1.718616
47  (Spanish Brunch)    (Coffee)            0.017761            0.470363  0.010533    0.593023  1.260777  0.002179    1.301393
35       (Medialuna)    (Coffee)            0.060409            0.470363  0.033664    0.557265  1.184754  0.005250    1.196283
39          (Pastry)    (Coffee)            0.084263            0.470363  0.045333    0.537990  1.143775  0.005698    1.146375
1        (Alfajores)    (Coffee)            0.035523            0.470363  0.019104    0.537791  1.143351  0.002395    1.145880
32           (Juice)    (Coffee)            0.037691            0.470363  0.020033    0.531507  1.129992  0.002305    1.130511
40        (Sandwich)    (Coffee)            0.070322            0.470363  0.036762    0.522761  1.111397  0.003685    1.109792
43           (Scone)    (Coffee)            0.033767            0.470363  0.017451    0.516820  1.098766  0.001569    1.096146
23            (Cake)    (Coffee)            0.101714            0.470363  0.052458    0.515736  1.096463  0.004615    1.093694
29         (Cookies)    (Coffee)            0.053181            0.470363  0.027365    0.514563  1.093969  0.002351    1.091051
31   (Hot chocolate)    (Coffee)            0.057001            0.470363  0.028294    0.496377  1.055305  0.001483    1.051652
21         (Brownie)    (Coffee)            0.039137            0.470363  0.018897    0.482850  1.026546  0.000489    1.024144
14          (Pastry)     (Bread)            0.084263            0.320322  0.027571    0.327206  1.021490  0.000580    1.010232
37          (Muffin)    (Coffee)            0.037691            0.470363  0.017968    0.476712  1.013498  0.000239    1.012133
```

Heatmap



Findings

- For Coffee, the following items are often bought together: Alfajores Brownie, Cake, Cookies, Hot Chocolate, Juice, Medialuna, Muffin, Pastry, Sandwich, Scone, Spanish Brunch, and Toast.
- For Bread, it is always bought with Pastry.
- For Tea, it is always bought with Cake.

These findings provide much concentrated view for the Top 3 best-selling products. Above products are the best combination of products according to the best Lift and Confidence at 20% and 1.0 respectively.

4.5. Recommendations

4.5.1. Introduce set breakfast or lunch

BreadBasket should offer set breakfast or lunch that allows customers to have something to eat with something to drink. The price of a set should lead to customers paying lower than paying for the items *ala carte*. This is to encourage more sales.

4.5.2. Slight adjustment in prices

Coffee is the most popular choice of drink, hence sets with coffee should be priced a little higher. Since Cake and Hot Chocolate, are bought almost together every time (high lift) though not frequent (low confidence), therefore another set can be introduced at a lower price than buying ala carte.

4.5.3. Mix n' Match

Introduce Mix n' Match whereby customers get to choose from an array of pastry and bread, pick a certain number, and price it competitively to encourage buying more than one at once. For example, because Bread and Pastry always get bought together, customers tend to want to have a selection of one or more combination at a lower price than buying them *ala carte*.

Chapter 5: Conclusion

In conclusion, the application provides a user-friendly interface for users to input records (either typing them using keyboard or upload a file path), select menu, input threshold values correctly, and displaying tables, charts, and heatmaps intuitively. Using BreadBasket dataset obtained from Kaggle, the application has proven its capability to read thousands of records, generate frequent itemsets using association analysis algorithms (Apriori and FP Growth) and association rules using different metrics (support, confidence, lift), and displaying charts and heatmaps. The program utilises the robust libraries in data science and analysis tools provided by the Python community. Ultimately, the program serves it purpose to show the correlations between products and lead to decision-making based on the findings and recommendations.

In my opinion, though the program has met its objective of performing association analysis on large datasets, its user interface may not be suitable to its target users who are not of computing background. The program is still limited to a command line program. In the future, the user interface can be enhanced using Tkinter which provides a framework for buttons, icons, and input fields. With a nice user interface, the program can be commercialised and serve its intended purpose of helping retailers make more informed decisions.

References

Aggarwal, C. C., Bhuiyan, M. A., & Hasan, M. A. (2014, August 30). Frequent Pattern Mining Algorithms: A Survey. *Frequent Pattern Mining*, 19 – 64. doi:10.1007/978-3-319-07821-2_2

Castillo, D. (2022, August 25). *What is transactional data and how can you leverage it?* From Lytics: https://www.lytics.com/blog/what-is-transactional-data-and-how-can-you-leverage-it/

Chee, C.-H., Jaafar, J., Aziz, I. A., Hasan, M. H., & Yeoh, W. (2018, March 24). Algorithms for frequent itemset mining: a literature review. *Artificial Intelligence Review, 52*, 2603 – 2621. doi:10.1007/s10462-018-9629-z

Getz, A. (2011, June 2). *Types of Enterprise Data (Transactional, Analytical, Master)*. From BI / DW Insider: https://bi-insider.com/posts/types-of-enterprise-data-transactional-analytical-master/

Jamsheela, O., & Raju, G. (2015). Frequent itemset mining algorithms: A literature survey. *2015 IEEE International Advance Computing Conference (IACC)* (pp. 1099 - 1104). Bangalore: IEEE. doi:10.1109/IADCC.2015.7154874

Matplotlib. (n.d.). *Matplotlib: Visualization with Python*. From Matplotlib: https://matplotlib.org/

NumPy. (n.d.). *NumPy - The fundamental package for scientific computing with Python*. From NumPy: https://numpy.org/about/

pandas. (n.d.). *pandas - Python Data Analysis Library*. From pandas: https://pandas.pydata.org/

Raj, S., Ramesh, D., & Sethi, K. K. (2021). A Spark-based Apriori algorithm with reduced shuffle overhead. *The Journal of Supercomputing, 77*, 133 – 151. doi:10.1007/s11227-020-03253-7

Raschka, S. (2018, April 22). MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *Journal of Open Source Software, 3*(24), 638. doi:10.21105/joss.00638

Raschka, S. (2022). *apriori: Frequent itemsets via the Apriori algorithm*. From Github.io: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

Raschka, S. (2022). *fpgrowth: Frequent itemsets via the FP-growth algorithm*. From Github.io: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/fpgrowth/

Seaborn. (n.d.). *Seaborn: statistical data visualization*. From Seaborn: https://seaborn.pydata.org/

Simplilearn. (2022, August 8). *How to Install PyCharm IDE: Everything You Need to Know*. From Simplilearn: https://www.simplilearn.com/tutorials/python-tutorial/pycharm