MCSD2123
Assignment 1

NAME: HO ZHI HOONG
MATRIC NO: MCS221002

SECTION 1: Exploratory Data Analysis (EDA)

Dataset_A has been imported into Jupyter Notebook and saved as a data frame named 'data', as illustrated in Figure 1.1. The data frame comprises 13951 rows and 3 columns.

```
data
```

	Member_number	Date	itemDescription	Year	Month
0	1808	2015-07-21	tropical fruit	2015	7
1	2552	2015-01-05	whole milk	2015	1
2	2300	2015-09-19	pip fruit	2015	9
3	1187	2015-12-12	other vegetables	2015	12
4	3037	2015-02-01	whole milk	2015	2
...	...	...	...	...	...
19346	3478	2015-04-02	rolls/buns	2015	4
19347	2295	2015-12-28	pip fruit	2015	12
19348	1833	2015-07-24	dessert	2015	7
19349	3307	2015-03-03	other vegetables	2015	3
19350	3562	2015-03-18	salty snack	2015	3

19351 rows × 5 columns

Figure 1.1: dataset

Figure 1.2 displays the data type of each column in the 'data' data frame and confirms the absence of any missing values. Furthermore, Table 1.1 below provides the descriptions of the columns. "Figure 1.3 below presents the number of distinct values in each column. It reveals that there are 728 unique values in the 'Date' column. Therefore, it is essential to verify the timeframe covered by this dataset.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19351 entries, 0 to 19350
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Member_number    19351 non-null  int64 
 1   Date             19351 non-null  object
 2   itemDescription  19351 non-null  object
dtypes: int64(1), object(2)
memory usage: 453.7+ KB
```

Figure 1.2

Table 1.1: Description of columns

Column Name	**Descriptions**
'Member_number'	identification number of customers who purchased the item
'Date'	transaction date
'itemDescription'	item sold

```
data.nunique()

Member_number      3873
Date                728
itemDescription     163
dtype: int64
```

Figure 1.3

Figure 1.4 shows that this dataset has daily transaction records from the years 2014 and 2015, with the majority of records being from 2015. The item sold in each month is also can be visualize directly from this. The highest number of items were sold in August 2015, while the lowest number of items were sold in March 2014.

```
#number of itmes sold in each month in each year.
data['Month'] = data['Date'].dt.month
data[['Year','Month']].value_counts()

Year  Month
2015  8        1137
      1        1093
      5        1086
      11       1076
      7        1058
      3        1029
      6        1020
      4        1008
      10       1000
      9         970
      2         951
      12        923
2014  5         616
      8         611
      10        610
      7         597
      6         597
      4         586
      1         579
      12        576
      9         576
      11        568
      2         546
      3         538
dtype: int64
```

Figure 1.4

By examining the distinct values of each column, we can infer that the dataset reflects a many-to-many relationship between customers and item descriptions, where each row corresponds to a specific item purchased by a specific customer on a particular date. Figure 1.5 below provides evidence for this, as the last two rows display that two items were bought by the same customer on the same date, but each item is recorded in a separate row.

```
data.sort_values(by=['Member_number','Date']).head(5)
```

	Member_number	Date	itemDescription	Year	Month
13331	1000	2014-06-24	whole milk	2014	6
4843	1000	2015-03-15	sausage	2015	3
8395	1000	2015-03-15	whole milk	2015	3
1629	1000	2015-05-27	soda	2015	5
17778	1000	2015-05-27	pickled vegetables	2015	5

Figure 1.5

Figure 1.6 displays a bar graph that shows the top 10 most frequent purchases items.

```
plt.figure(figsize=(7,5))
freq = data['itemDescription'].value_counts().head(10)
freq.plot(kind='barh')
plt.title('Top 10 Most Frequent Purchases', fontsize=16)
plt.xlabel('Frequency', fontsize=10)
plt.ylabel('Item', fontsize=10)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.show()
```

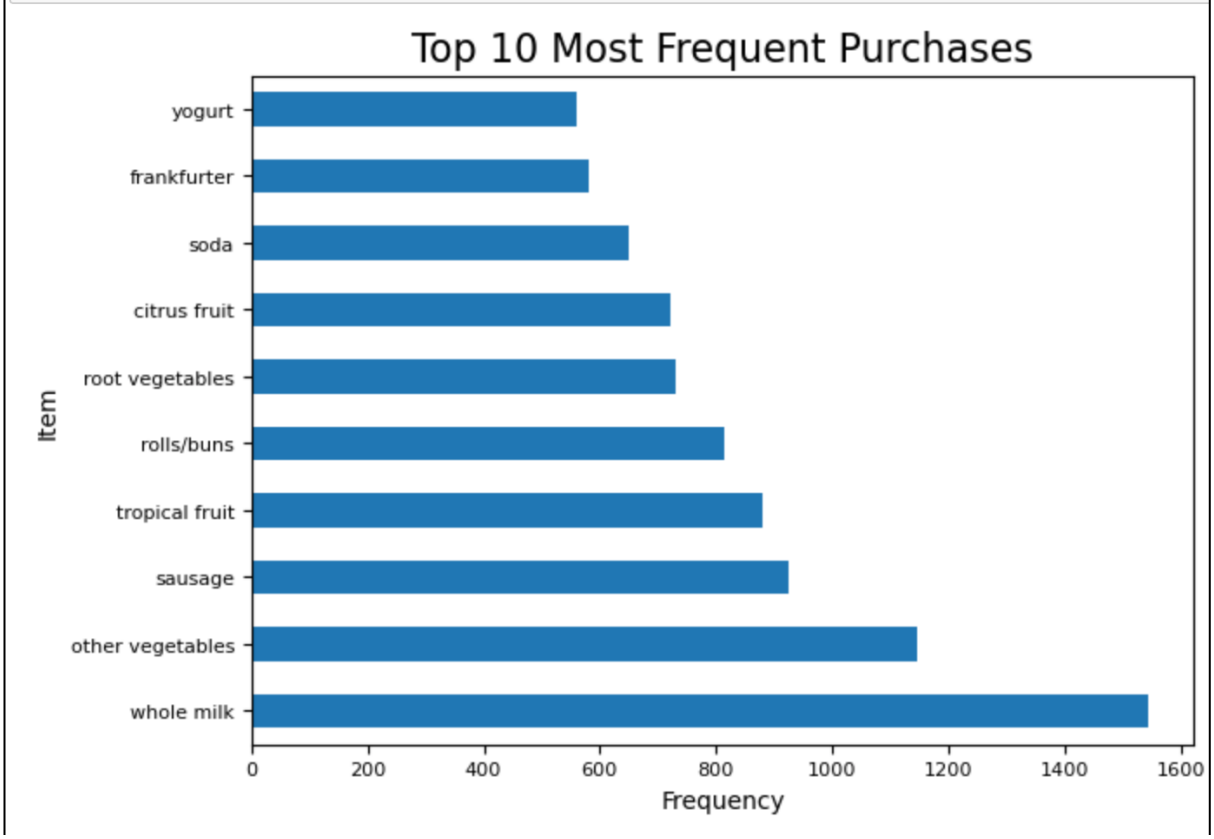

Figure 1.6

SECTION 2: Preparation of Dataset

To prepare the dataset for association rule mining, the first step is to group the data by 'Date' and 'Member_number' and store the purchased items for each customer on each day in list format, as illustrated in Figure 2.1 below. A new column called 'basket' is created to store the list of items bought. This grouped dataframe comprises 13971 rows and 3 columns, with each row representing a single transaction.

```
group=data.groupby(['Date', 'Member_number'],
                   axis=0)['itemDescription'].apply(list
                    ).reset_index(name='basket')
print(group.info())
group
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13971 entries, 0 to 13970
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           13971 non-null  datetime64[ns]
 1   Member_number  13971 non-null  int64
 2   basket         13971 non-null  object
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 327.6+ KB
None
```

	Date	Member_number	basket
0	2014-01-01	1249	[citrus fruit]
1	2014-01-01	1381	[curd]
2	2014-01-01	1440	[other vegetables]
3	2014-01-01	1659	[specialty chocolate]
4	2014-01-01	1789	[hamburger meat]
...	...	...	...
13966	2015-12-30	3738	[onions]
13967	2015-12-30	3971	[brown bread, bottled beer]
13968	2015-12-30	4058	[domestic eggs, cream cheese]
13969	2015-12-30	4565	[canned beer, canned beer]
13970	2015-12-30	4863	[dessert, curd]

13971 rows × 3 columns

Figure 2.1

The next step involves using the TransactionEncoder function from the mlxtend library to perform one-hot encoding on the 'basket' column. This function takes a list of lists, where each inner list represents the items in a transaction, and returns a one-hot encoded dataset. In this case, the function takes the 'basket' column and encodes it into a dataset where each row represents a transaction and each column represents an item, as shown in Figure 2.2. The number of columns in the encoded dataset corresponds to the number of distinct values in the 'itemDescription' column. Finally, the encoded data is converted into a Pandas DataFrame, and the columns are named using the unique item names obtained from the encoder. This encoded dataset is now ready to be utilized in association rule mining algorithms, Apriori.

```
from mlxtend.preprocessing import TransactionEncoder

encoder = TransactionEncoder()
encoded_data = encoder.fit(group['basket']).transform(group['basket'])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.columns_)
encoded_df
```

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	turkey	vinegar	waffles	whipped/sour cream	whisky	white bread
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13966	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
13967	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
13968	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
13969	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
13970	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False

13971 rows × 163 columns

Figure 2.2

SECTION 3: Methodology

The Apriori algorithm was used as the methodology for mining association rules in this study. This algorithm is popular in data mining for identifying frequent itemsets in a dataset. It works by generating frequent itemsets from the given dataset and then forming rules based on minimum support and minimum confidence thresholds.

To perform the association rules mining, the Apriori and association rules functions were imported from the mlextend library. The required metrics for these two functions are support, confidence, and lift thresholds. The choice of support threshold involves a trade-off between the number of generated rules and their relevance. A higher support threshold will generate fewer rules, but they will be more significant. Conversely, a lower support threshold will generate more rules, but they may be less relevant. Therefore, a loop function was utilized to determine the number of rules generated at each specific support threshold, which was set to a range from 0.00005 to 0.01, as shown in Figure 3.1.

In addition, the association rules function was applied with a lift metric, with a threshold set at 1. The lift metric is more significant than the confidence metric because it measures the strength of association between itemsets, whereas confidence can be misleading when the individual item purchase frequencies are already high. The lift metric tells us how much more likely it is to buy two items together than if they were purchased independently, providing more information about the strength of their association. Therefore, lift metric is used in this association rules mining.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

count_ms = []
count_len = []

def find_rule(msupp):
    frequent_itemset_ap = apriori(encoded_df, min_support=msupp, use_colnames=True)
    rules = association_rules(frequent_itemset_ap, metric='lift', min_threshold=1)
    count_ms.append(msupp)
    count_len.append(len(rules))
    return rules
# Set the initial minimum support threshold
msupp = 0.00005

# Call the app() function with increasing values of msupp until a maximum value is reached
max_msupp = 0.01
while msupp <= max_msupp:
    find_rule(msupp)
    msupp += 0.00005
```

Figure 3.1

The loop function produces a list of the number of rules based on the support threshold used in the Apriori algorithm, as displayed in Figure 3.2. This list makes it easy to visualize and determine the appropriate support threshold based on the desired number of rules. The higher the support threshold ('min_supp'), the fewer the number of rules generated ('num_of_rule'). If the support threshold is too high, it may not yield any rules because all rules fail to meet the threshold and are filtered out of the function. Therefore, based on the list in Figure 3.2, the maximum support threshold that results in at least one rule is 0.0011. Setting the support threshold higher than this value will result in no rules.

```
count_rules = pd.DataFrame({'min_supp':count_ms,
                            'num_of_rule':count_len})
count_rules[count_rules['num_of_rule']!=0]
```

	min_supp	num_of_rule
0	0.00005	22308
1	0.00010	1332
2	0.00015	424
3	0.00020	424
4	0.00025	168
5	0.00030	80
6	0.00035	80
7	0.00040	38
8	0.00045	20
9	0.00050	20
10	0.00055	16
11	0.00060	10
12	0.00065	8
13	0.00070	8
14	0.00075	8
15	0.00080	6
16	0.00085	6
17	0.00090	6
18	0.00095	6
19	0.00100	6
20	0.00105	4
21	0.00110	2

Figure 3.2

SECTION 4: Result and Interpretation

To generate 10 rules, a support threshold of 0.006 was used in the Apriori algorithm function. This threshold was inputted into the function, along with a lift metric threshold of 1 for the association rules function. The resulting 10 rules are displayed in Figure 4.1.

```
rules_df = find_rule(0.0006)
rules_df
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(beverages)	(sausage)	0.010307	0.064634	0.000787	0.076389	1.181871	0.000121	1.012727	0.155486
1	(sausage)	(beverages)	0.064634	0.010307	0.000787	0.012182	1.181871	0.000121	1.001898	0.164517
2	(pork)	(brown bread)	0.037005	0.014029	0.000644	0.017408	1.240862	0.000125	1.003439	0.201567
3	(brown bread)	(pork)	0.014029	0.037005	0.000644	0.045918	1.240862	0.000125	1.009342	0.196870
4	(citrus fruit)	(butter)	0.050891	0.015604	0.001002	0.019691	1.261913	0.000208	1.004169	0.218681
5	(butter)	(citrus fruit)	0.015604	0.050891	0.001002	0.064220	1.261913	0.000208	1.014244	0.210842
6	(sausage)	(curd)	0.064634	0.015890	0.001074	0.016611	1.045389	0.000047	1.000733	0.046419
7	(curd)	(sausage)	0.015890	0.064634	0.001074	0.067568	1.045389	0.000047	1.003146	0.044120
8	(grapes)	(whole milk)	0.010164	0.108153	0.001145	0.112676	1.041825	0.000046	1.005098	0.040558
9	(whole milk)	(grapes)	0.108153	0.010164	0.001145	0.010589	1.041825	0.000046	1.000430	0.045014

Figure 4.1

The rules shown in Figure 4.1 contain some duplicate combinations appearing as {A:B} and {B:A}. This is because both combinations have the same lift value, as determined by the lift formula. As a result, although there are 10 rules displayed, there are actually only 5 unique combinations as shown in Table 4.1.

$$Lift(\{A\} \rightarrow \{B\}) = \frac{supp(\{A, B\})}{supp(\{A\}) \times supp(\{B\})}$$

Table 4.1

Index	**Combinations**	
0 & 1	Beverages	Sausage
2 & 3	Pork	Brown bread
4 & 5	Citrus fruit	Butter
6 & 7	Sausage	Curd
8 & 9	Grapes	Whole milk

The lift metric is a ratio of the observed support of both the antecedent and consequent items, compared to the expected support if they were independent. A value of 1 indicates no association between the antecedent and consequent items.

If the lift value is greater than 1, it suggests a positive association between the antecedent and consequent items, meaning they are more likely to be purchased together than separately. As the lift value increases, so does the strength of the association between the items. This information can be used to make targeted recommendations or promotions to customers, based on their purchasing history.

On the other hand, if the lift value is less than 1, it indicates a negative association between the antecedent and consequent items. In this case, the items are less likely to be purchased together than independently. Negative association can be useful in identifying items that should not be recommended together, such as incompatible products or items that may conflict with each other.

For instance, in the case of the first two combinations in the rules data frame, {beverages: sausage} and {sausage: beverages}, both combinations have a lift value of 1.18, which suggests that the probability of purchasing sausage is 1.18 times higher when the customer also buys beverages, compared to the probability of buying sausage without considering the purchase of beverages. It is important to note that lift does not represent a percentage probability, but rather a measure of the strength of association between the antecedent and consequent items. In this case, a lift of 1.18 indicates a positive association between "beverages" and "sausage."

Using association rule recommendation, it would be more effective to recommend "sausage" to a customer after they have added "beverages" to their shopping cart, rather than directly recommending "sausage." The same applies to the combination {sausage: beverages}.

The graph approach is a useful technique to identify the most interesting rules from a set of rules generated by the association rules mining. In this case, three different graphs were plotted (Figure 4.2, Figure 4.3, and Figure 4.4) to analyze the relationship between lift, confidence, and support for the 10 rules in the rules dataframe.

By examining the graphs, it becomes easier to identify the rules that have high lift, high confidence, and high support, which are the most interesting rules that should be focused on. The interested rules are indicated by their indexes at the top right of each graph.

Table 4.2 summarizes the combinations of the interested rules, with the exception of the combinations of rules 2 and 3, which involve the items "pork" and "brown bread". The graph approach recommends all other combinations of rules for further analysis.In summary, the graph approach is a powerful tool to visualize the relationships between different metrics and select the most interesting rules for further analysis.

Table 4.2

Index	**Combinations**		**Graphs**			**Graphs recommended**
			Lift vs confidence	**Lift vs support**	**Support vs confidence**	
0 & 1	Beverages	Sausage	•			•
2 & 3	Pork	Brown bread				
4 & 5	Citrus fruit	Butter	•	•	•	•
6 & 7	Sausage	Curd			•	•
8 & 9	Grapes	Whole milk			•	•

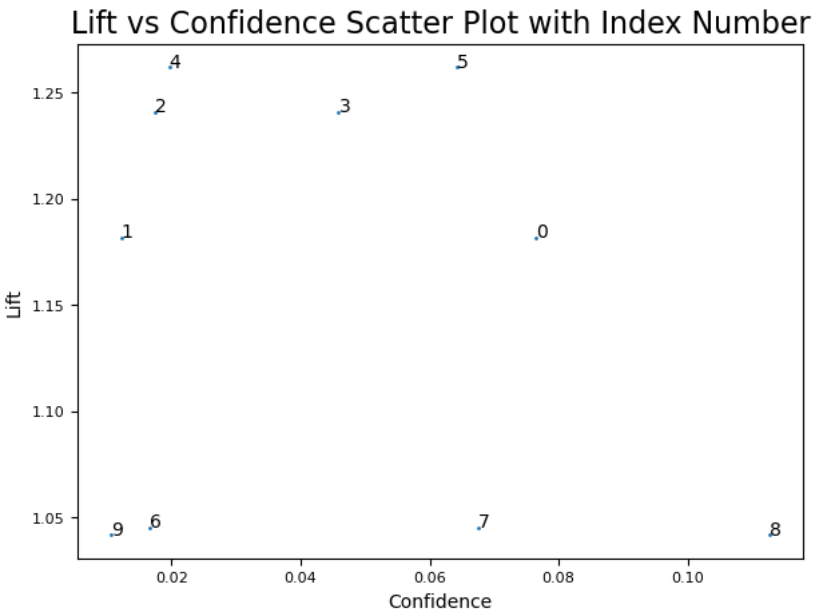

Figure 4.2

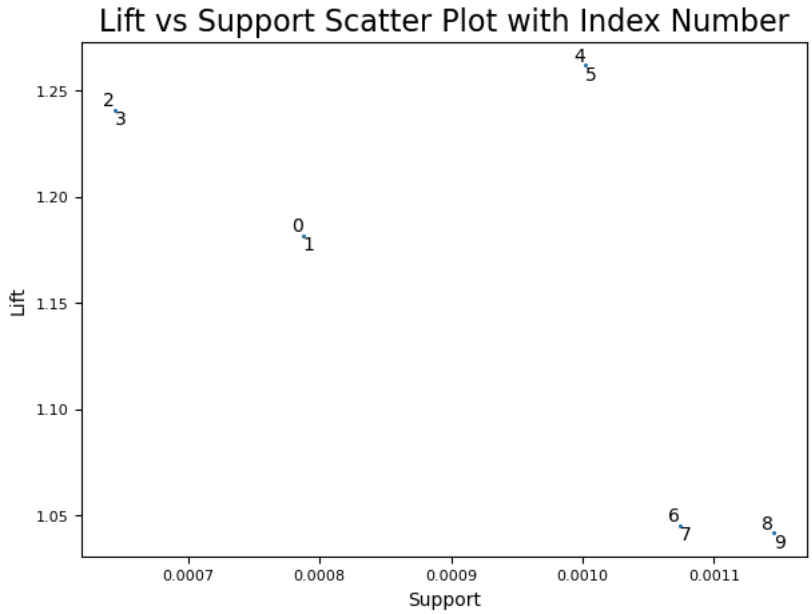

Figure 4.3

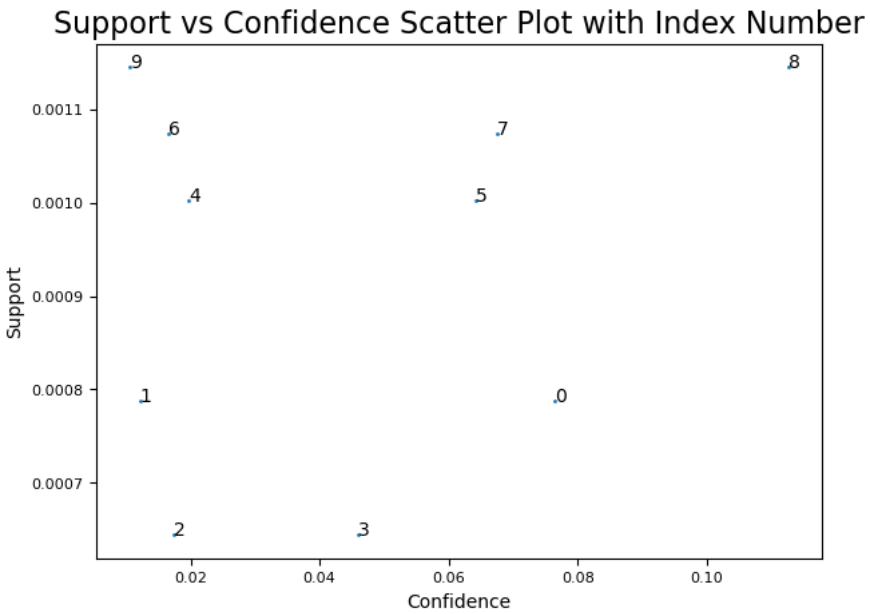
Support vs Confidence Scatter Plot with Index Number
Support
Confidence
0.0011
0.0010
0.0009
0.0008
0.0007
0.02
0.04
0.06
0.08
0.10
9
8
6
7
4
5
1
0
2
3

Figure 4.4