MCSD2123
Assignment 1

NAME: FAN CHIN WEI
MATRIC NO: MCS221024

SECTION 1: Exploratory Data Analysis (EDA)

In this work, EDA is used to analyze and understand the dataset before going with the association. The python code reads a CSV files named "Dataset_B.csv" into a pandas DataFrame called "data".

```
# Read the dataset and save the data in dataframe

data = pd.read_csv("Dataset_B.csv")
store_sales = pd.DataFrame(data)

# Print first 5 rows of data
store_sales.head()
```

	Member_number	Date	itemDescription
0	3562	18-03-2015	salty snack
1	3145	16-11-2015	cake bar
2	3595	17-12-2015	whole milk
3	4934	17-03-2015	other vegetables
4	3386	3/2/2015	yogurt

The data type of each column in the "store_sales" dataframe are displayed as below.

```
: # Display data type for each column

  store_sales.dtypes
```

```
: Member_number      int64
  Date              object
  itemDescription   object
  dtype: object
```

The dataset B contains 19415 rows and 3 columns as shown in below.

```
: # Display dimension of store_sales

  store_sales.shape
```

```
: (19415, 3)
```

The dataset has no null values as illustrate in below.

```
# Check null values in the dataset

store_sales.isnull().sum()
```

```
Member_number      0
Date               0
itemDescription    0
dtype: int64
```

In the other words, the dataset comprises 164 unique items and 3814 member numbers. The occurrence of each item is counted by using value_counts function.

```
# Print the total of unique items

len(store_sales.itemDescription.unique())
```

```
164
```

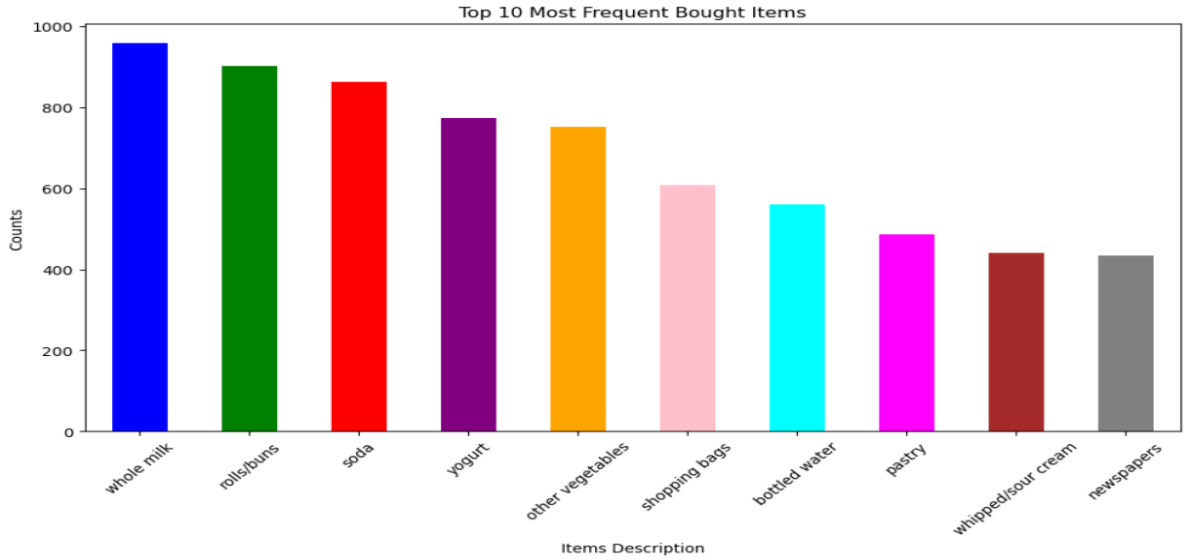
```
# Print the total of member number

len(store_sales.Member_number.unique())
```

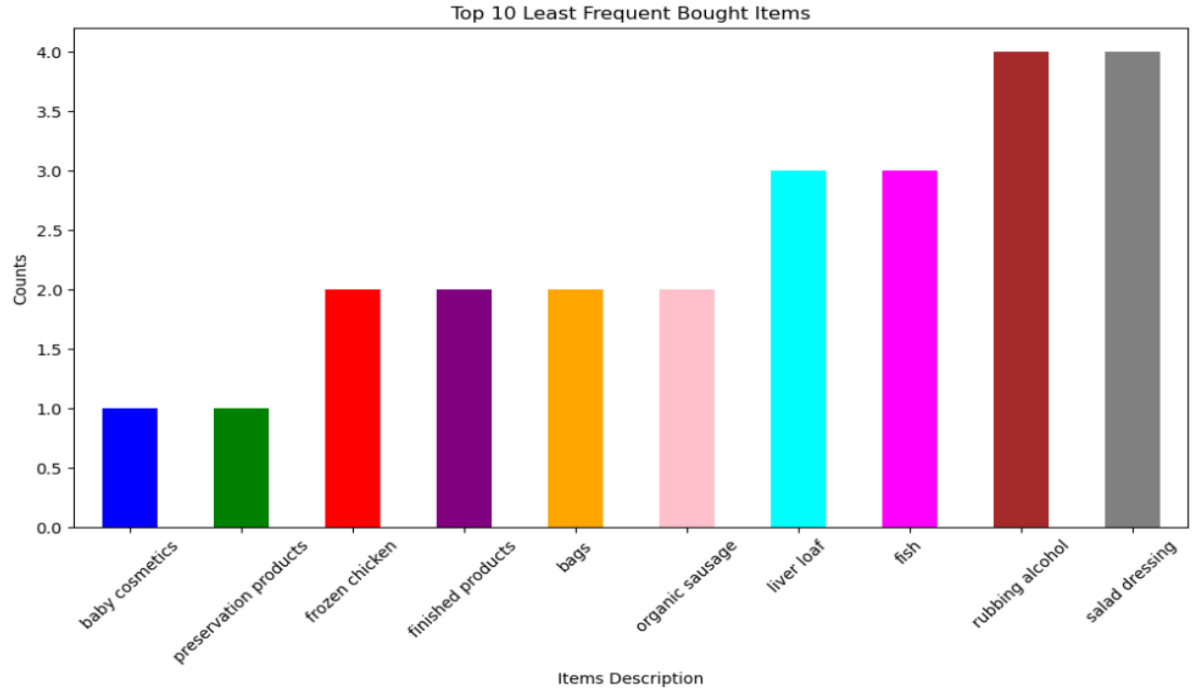
```
3814
```

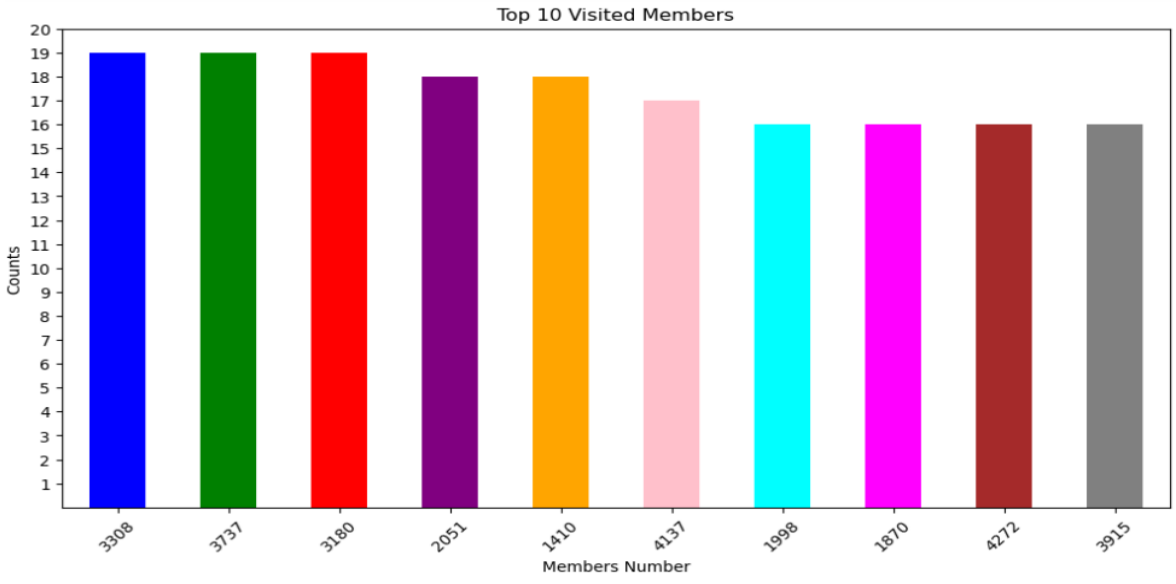
```
# Print out the total of each items in itemsDescription column

items_sales = store_sales["itemDescription"].value_counts()
items_sales
```

```
whole milk               957
rolls/buns               901
soda                     863
yogurt                   774
other vegetables         752
                        ...
organic sausage            2
frozen chicken             2
bags                       2
preservation products      1
baby cosmetics             1
Name: itemDescription, Length: 164, dtype: int64
```

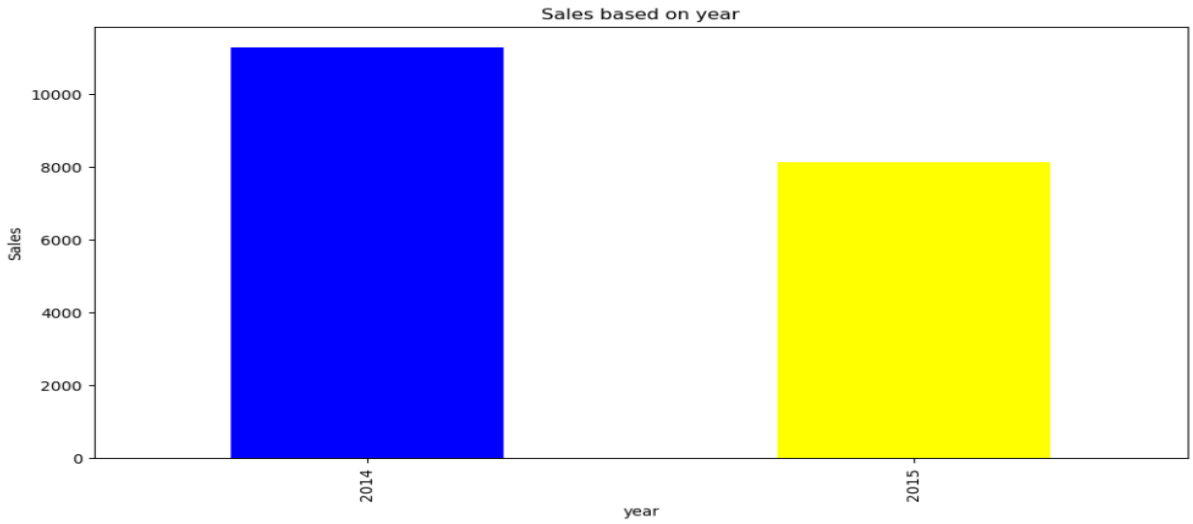
Furthermore, various visualization techniques, including bar graphs, pie charts, and tree maps, have been employed to represent the dataset. Specifically, a bar graph has been utilized to visualize the top 10 most and least frequently purchased items by customers and top 10 visited member. This visualization provides a clear and concise representation of the popularity distribution among the items, enabling easy identification of both the most and least favored products based on customer buying patterns. Moreover, it appears that the goal is to visualize and gain insights into the visit counts for the top 10 visited members. The bar graph visually represents the distribution of visit counts across these members, with each bar corresponding to a member. By examining the heights of the bars, viewers can quickly identify which members are the most frequently visited, providing valuable insights into user engagement or popularity within the dataset.


Top 10 Most Frequent Bought Items
Counts
1000
800
600
400
200
0
whole milk
rolls/buns
soda
yogurt
other vegetables
shopping bags
bottled water
pastry
whipped/sour cream
newspapers
Items Description

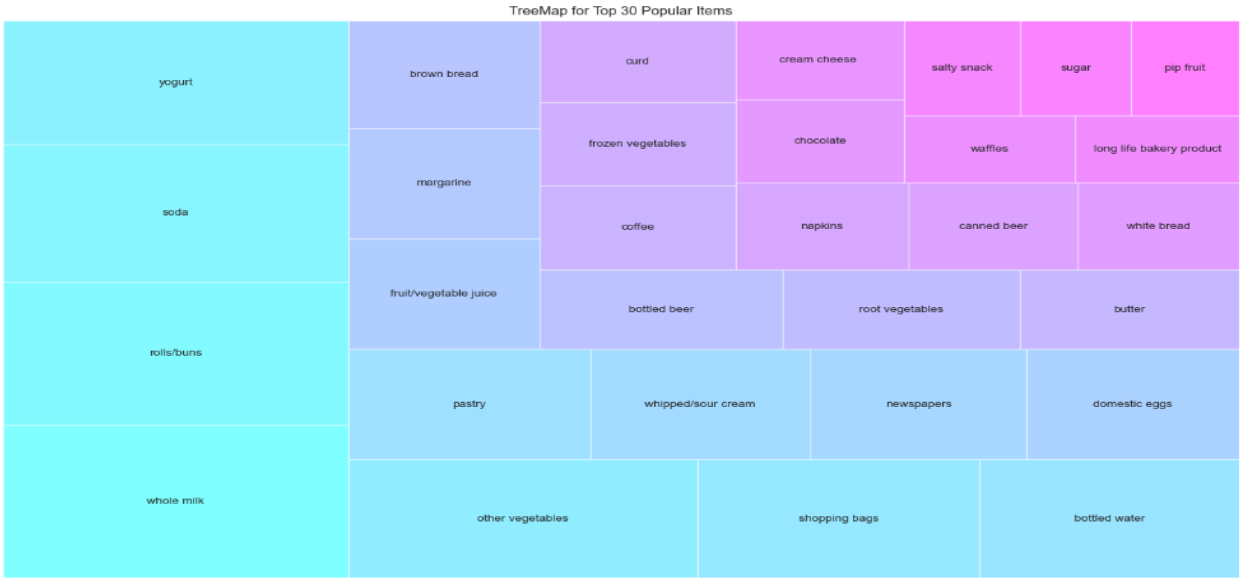

Top 10 Least Frequent Bought Items
Counts
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
baby cosmetics
preservation products
frozen chicken
finished products
bags
organic sausage
liver loaf
fish
rubbing alcohol
salad dressing
Items Description


Top 10 Visited Members
Counts
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
3308
3737
3180
2051
1410
4137
1998
1870
4272
3915
Members Number

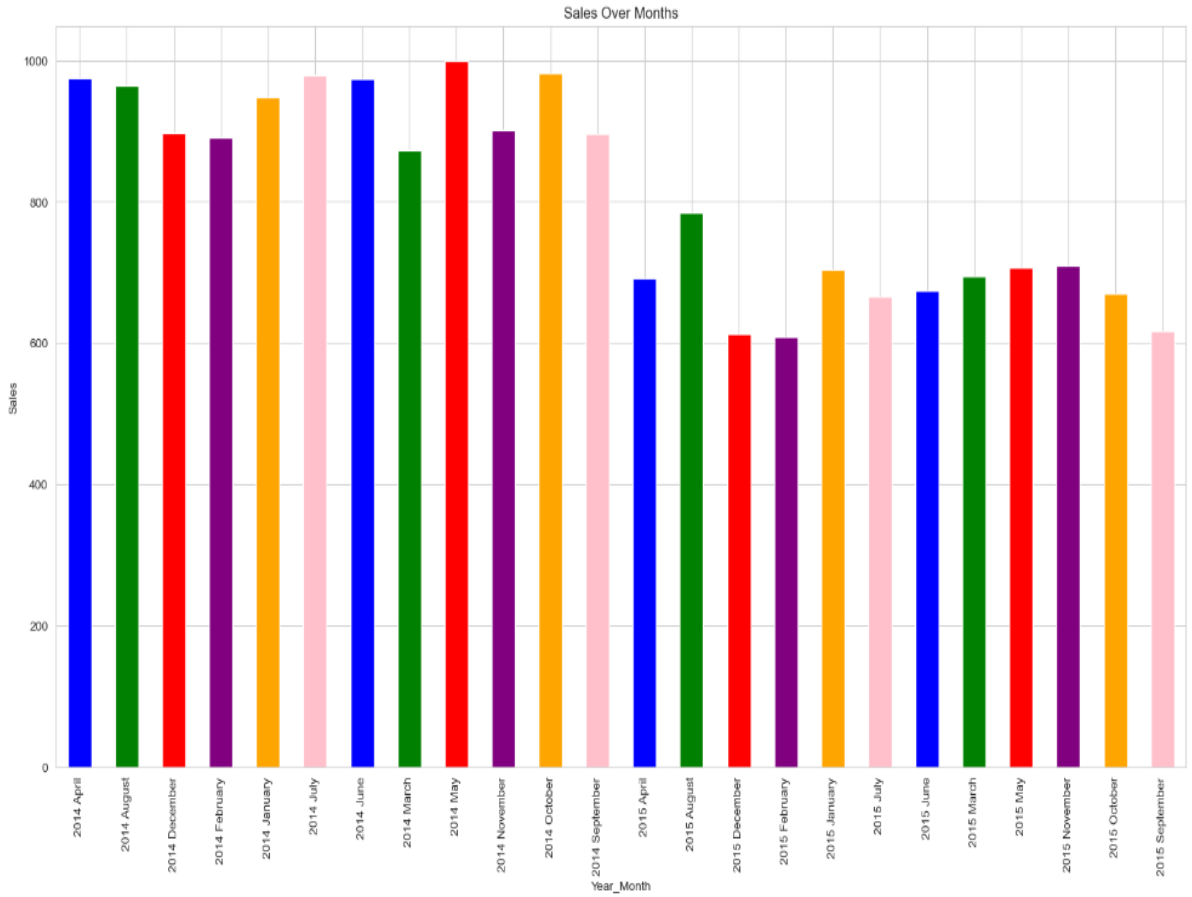
This Python code uses the matplotlib library to create a bar graph depicting sales based on the year. The figure size is set to 12x6 inches. The “store_sales” DataFrame is grouped by the “year” column, and the count of item descriptions for each year is then plotted as a bar graph with blue and yellow bars representing different years. The y-axis is labeled as sales and the graph's title is set as Sales based on year. This visualization provides an overview of how sales are distributed across different years, allowing for quick insights into potential trends or variations in sales over time.



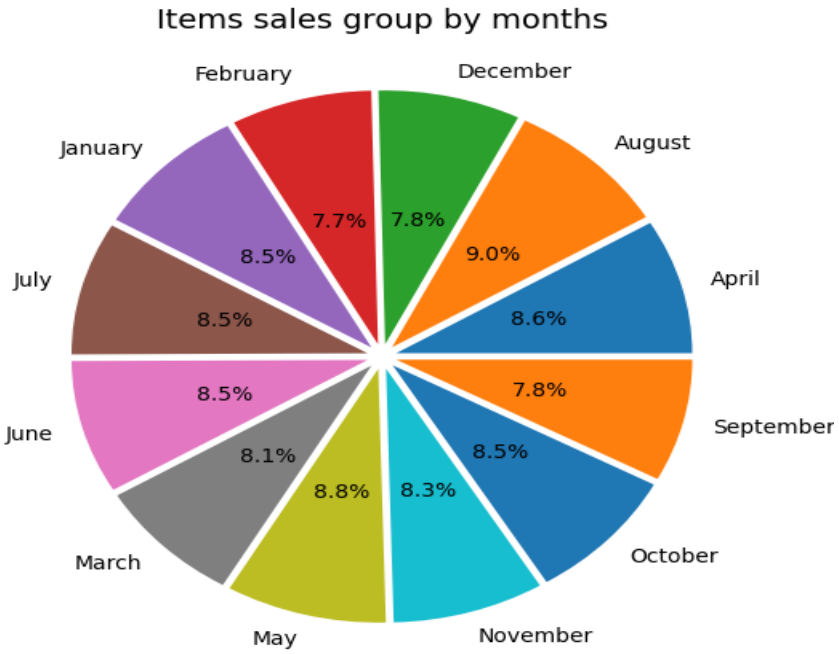
The “squarify” and “seaborn” libraries are used to create a treemap visualization for the top 30 popular items based on their counts in the “store_sales” dataset. The variable, x is assigned the value counts of item descriptions for the top 30 items, converted to a DataFrame. The treemap is then titled as TreeMap for Top 30 Popular Items, and the axis is turned off for better visualization. Overall, this code produces an informative treemap visualization that visually represents the relative popularity of the top 30 items in the dataset.



In addition, this visualization offers insights into the monthly distribution of sales, potentially revealing patterns or trends over the specified time period.



Moreover, this visualization offers a concise representation of the relative contribution of each month to the overall item sales.



Furthermore, the resulting “cleaned_store_sales” pivot table provides a structured overview of item counts for each month, with a column indicating the item with the highest count in each month. This type of data processing is beneficial for analyzing and exploring trends in item sales over time.

itemDescription	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	vinegar	waffles	whipped/sour cream	whisky	white bread	w
Month_name																	
April	3.0	14.0	0.0	0.0	0.0	0.0	11.0	0.0	1.0	7.0	...	1.0	16.0	34.0	1.0	21.0	
August	5.0	12.0	3.0	1.0	0.0	0.0	7.0	1.0	5.0	4.0	...	3.0	15.0	21.0	0.0	23.0	
December	2.0	14.0	1.0	1.0	0.0	0.0	4.0	1.0	3.0	8.0	...	3.0	26.0	40.0	1.0	18.0	
February	3.0	7.0	2.0	5.0	0.0	1.0	5.0	0.0	5.0	7.0	...	1.0	17.0	34.0	0.0	21.0	
January	7.0	15.0	2.0	2.0	1.0	0.0	11.0	0.0	3.0	11.0	...	11.0	22.0	44.0	1.0	24.0	
July	5.0	18.0	2.0	1.0	0.0	0.0	10.0	3.0	4.0	5.0	...	2.0	15.0	32.0	0.0	23.0	
June	2.0	15.0	1.0	3.0	0.0	0.0	15.0	1.0	5.0	9.0	...	4.0	14.0	50.0	0.0	26.0	
March	5.0	18.0	3.0	2.0	0.0	1.0	4.0	0.0	5.0	8.0	...	4.0	12.0	30.0	0.0	11.0	
May	3.0	16.0	1.0	0.0	0.0	0.0	7.0	0.0	2.0	13.0	...	4.0	23.0	44.0	1.0	18.0	
November	1.0	12.0	0.0	0.0	0.0	0.0	10.0	1.0	2.0	8.0	...	5.0	14.0	34.0	0.0	29.0	
October	6.0	10.0	1.0	3.0	0.0	0.0	5.0	3.0	3.0	9.0	...	2.0	18.0	25.0	1.0	30.0	
September	3.0	14.0	3.0	3.0	0.0	0.0	9.0	2.0	5.0	11.0	...	2.0	17.0	52.0	1.0	12.0	

12 rows × 165 columns

In conclusion, the removing unnecessary columns in needed after EDA is completed. This is to focus on relevant columns or simplify the dataset for further analysis.

	Member_number	itemDescription	formatted_date
0	3562	salty snack	2015-03-18
1	3145	cake bar	2015-11-16
2	3595	whole milk	2015-12-17
3	4934	other vegetables	2015-03-17
4	3386	yogurt	2015-02-03

SECTION 2: Preparation of Dataset

To identify associations, it is crucial to discover frequent itemsets, representing tuples or subsets of items. In the provided dataset, individual products are distinct entities. Consequently, it is imperative to aggregate purchased items based on Member_number and Date. This signifies that the items bought on the same day by the same customer, appearing on a single receipt, must be grouped together. Therefore, transactions occurring on identical dates and associated with the same member will be consolidated as a unified group. The figure below shown that the store sales data is grouped by the member number and formatted date.

```
# Groups the DataFrame by 'Member_number' and 'formatted_date' and applying a lambda function to create a list of 'itemDescriptio
store_sales=store_sales.groupby(['Member_number','formatted_date'])['itemDescription'].apply(lambda x: list(x))
```

```
store_sales.head(10)
```

```
Member_number  formatted_date
1000           2014-06-24              [pastry, salty snack]
               2015-03-15       [semi-finished bread, yogurt]
               2015-11-25                 [hygiene articles]
1001           2014-02-07           [whole milk, rolls/buns]
               2014-12-12                           [soda]
               2015-01-20               [whipped/sour cream]
               2015-04-14                     [white bread]
               2015-05-02                           [curd]
1002           2014-02-09                 [other vegetables]
               2014-04-26                       [whole milk]
Name: itemDescription, dtype: object
```

Furthermore, the TransactionEncoder class from the mlxtend.preprocessing module is used to convert a list of transactions (presumably named as sales_records) into a one-hot encoded DataFrame. The resulting "encoded_df" DataFrame represents the transactions in a categorical format, where each column corresponds to a unique item, and each row corresponds to a transaction. The values are True or False, indicating whether a particular item is present in a transaction.

```
from mlxtend.preprocessing import TransactionEncoder
x = TransactionEncoder()
x_data = x.fit(sales_records).transform(sales_records)
encoded_df = pd.DataFrame(x_data,columns=x.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	wl w
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12753	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
12754	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
12755	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
12756	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa
12757	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	Fa

12758 rows × 164 columns

SECTION 3: Methodology

The methodology is conducted by using both algorithm such as Apriori and FP-growth. Understanding the Apriori and FP-growth algorithms is critical for effective association rule mining. Both techniques are often employed in transactional datasets to discover frequent itemsets. Furthermore, the Apriori algorithm generates candidate itemsets iteratively and prunes those that do not reach a predefined minimum support level. The minimal support level, which specifies the threshold for itemset inclusion, is the most important parameter in Apriori. While Apriori is good in identifying common itemsets, it may necessitate numerous passes over the data, resulting in longer execution times for large datasets. On the other hand, the FP-growth algorithm employs a divide-and-conquer technique, building a tree-like structure known as the FP-tree. When compared to Apriori, this technique considerably minimizes the need for numerous database scans. The minimal support threshold, similar to Apriori, is the primary parameter for FP-growth. In addition, FP-growth often outperforms in terms of execution time, especially when working with huge datasets, because it generates a compact data structure that allows for faster mining. Furthermore, in terms of efficiency, the FP-growth algorithm frequently surpasses Apriori, making it the favored choice in circumstances where computational speed is critical. The decision between Apriori and FP-growth is determined by parameters such as dataset size, complexity, and the trade-off between ease of implementation and computing performance. In summary, both the Apriori and FP-growth algorithms require a minimum support parameter, but FP-growth outperforms Apriori in terms of execution time, especially with large datasets, due to its efficient tree-based structure. The decision between these techniques is determined by the features of dataset and the trade-off between ease of implementation and computing performance.

In this study, choosing the suitable parameters is a key stage in developing the association model. Total transaction is 12758 rows. Based on the assumption from online source (link), suppose that itemsets were bought 3 times a month and the data in dataset were collected from January 2014 to December 2015 (24 months).

minimum support threshold = (3*24)/12758=0.006

The choice of a minimum support threshold of 0.006 is subjective and depends on the characteristics of your dataset and the specific goals of your analysis. Lower support thresholds will result in more itemsets being considered frequent. The result with minimum support threshold of 0.006 shown as below. The aim of this study is setting the minimum support to a

lower value is advisable to obtain frequent itemsets that include a minimum of two items in tuple. However, the result return none when the minimum support threshold is 0.006.

```
encoded_df1[(encoded_df1['itemset_lengths']>=2 )]
```

	support	itemsets	itemset_lengths

Based on the result, the resulting DataFrame (encoded_df1) will contain frequent itemsets along with their support values. The filtering step (encoded_df1['itemset_lengths'] >= 2) ensures that only rules with at least two items in the antecedent or consequent are considered. After applying these thresholds, you can interpret the rules as associations between items that occur together frequently in your dataset. The support value indicates the proportion of transactions that contain a particular itemset.

After that, we set the minimum support threshold below:

minimum support threshold = (2*24)/12758=0.004

The result return none when the minimum support threshold is 0.004 that illustrate as below.

```
# Check the length of the itemsets equal or more than 2
encoded_df1[(encoded_df1['itemset_lengths']>=2 )]
```

	support	itemsets	itemset_lengths

In addition, we set the minimum support threshold below:

minimum support threshold = (1*24)/12758=0.002

Based on the result, seven itemsets meet the minimum support threshold of 0.002, and these itemsets specifically have a length of 2. This requirement ensures that each itemset contains at least two items in a tuple that illustrate as below.

```
# Check the length of the itemsets equal or more than 2
encoded_df1[(encoded_df1['itemset_lengths']>=2 )]
```

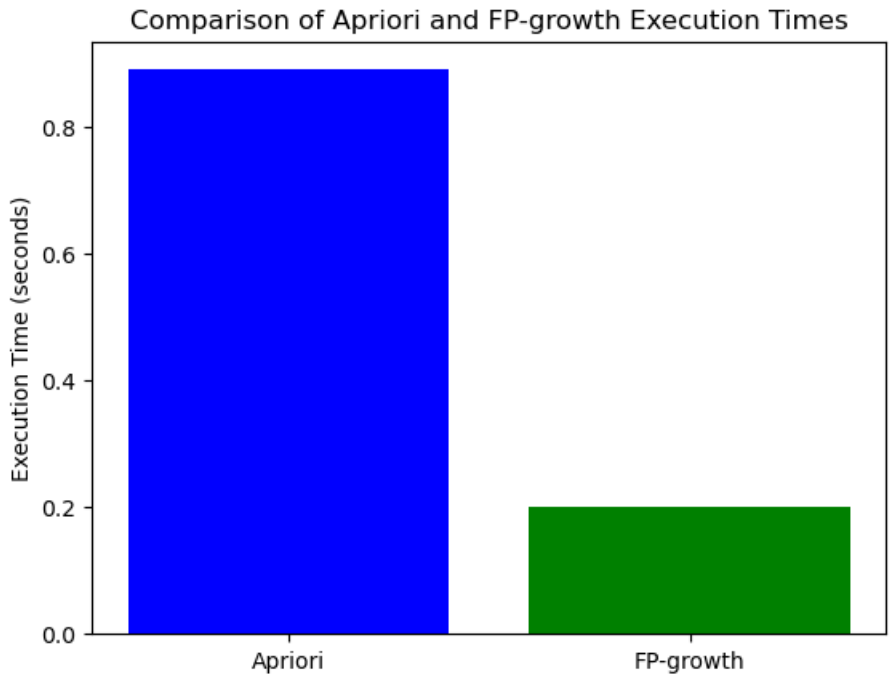
	support	itemsets	itemset_lengths
111	0.002430	(other vegetables, whole milk)	2
112	0.002273	(rolls/buns, shopping bags)	2
113	0.002351	(rolls/buns, soda)	2
114	0.002587	(rolls/buns, whole milk)	2
115	0.002351	(shopping bags, soda)	2
116	0.002508	(whole milk, soda)	2
117	0.002587	(yogurt, soda)	2

Furthermore, confidence measures the reliability of the rule. It is the probability of finding the consequent (output) in a transaction given that the transaction contains the antecedent (input). In this study, association rules are derived using the Apriori and FP-growth algorithms with specific parameter choices. Given the relatively low support values ranging from 0.000078 to 0.074 for each item in the dataset, and considering the dataset's substantial size of 12758 rows, a confidence threshold of 3% (0.03) has been set to identify meaningful associations. Additionally, the minimum length of itemsets has been specified as 2, focusing on uncovering patterns related to items frequently purchased together by customers. The Apriori and FP-growth algorithms are applied to extract association rules based on these criteria (shown in figure below). To ensure the accuracy of algorithms used, the same parameters used in Apriori algorithm will be applied in FP-growth algorithm.

```
: apriori_rules = association_rules(encoded_df1, metric="confidence", min_threshold=0.03)
apriori_rules
```

```
frequent_itemsets_fp=fpgrowth(encoded_df, min_support=0.002, use_colnames=True)
fp_rules = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=0.03)
fp_rules
```

Moreover, the bar graph calculates and visualises the execution time of both techniques. The FP-growth method takes less time to execute than the Apriori method. It shows that FP-growth processes the dataset faster than Apriori.



SECTION 4: Result and Interpretation

In this study, the analysis involved applying both the Apriori and FP-Growth algorithms to mine association rules from a dataset represented by the encoded_df DataFrame. For the Apriori algorithm, a minimum support threshold of 0.002 was chosen, indicating that only itemsets with a support of at least 0.2% were considered frequent. Additionally, rules with a confidence greater than or equal to 0.03 were extracted. The FP-Growth algorithm was then employed with a lower minimum support threshold of 0.002 to identify more patterns in the data (illustrated in the figures below).

The resulting association rules from both algorithms, stored in the apriori_rules and fp_rules DataFrames, provide insights into relationships between items. These rules are characterized by metrics such as support, confidence, and lift. Support indicates the proportion of transactions containing a specific itemset, confidence measures the conditional probability of the consequent given the antecedent, and lift quantifies the degree to which the items are more likely to be purchased together compared to if they were bought independently. Analysing the results includes thoroughly inspecting the created rules in order to identify important patterns and appreciate their significance for decision-making. Rules with high confidence indicate strong correlations, while lift values more than one indicate that products are more likely to be purchased together than would be predicted by chance. To validate the significance of the identified connections, domain knowledge and context must be included. Furthermore, the execution time in FP-growth is shorter than Apriori. This is because of the FP-growth algorithm requires less memory since it needs to scan the dataset twice.

Recommendations based on the analysis might include actionable insights for marketing strategies, product placement. Through association rule mining analysis, the discovered rules can provide valuable recommendations to store owners for optimizing the placement of items to enhance sales. High-confidence rules, indicative of robust associations between items, can inform the creation of appealing product groupings or displays. For example, if certain items consistently co-occur in transactions, situating them in close proximity within the store may prompt customers to purchase both items. Furthermore, lift values exceeding 1 highlight item groups more likely to be purchased together than if selected independently, offering insights into synergies aligning with customer preferences.

In summary, I would like to suggest the store owner might strategically position these items near each other to leverage this connection. Additionally, targeted promotions or discounts for these item pairs could be implemented to further encourage customers to make joint purchases. By implementing these insights in product placement and promotions, the store can enhance the shopping experience, enhance customer satisfaction, and ultimately drive sales. The figures below shown the Apriori and FP-growth results. The results obtained from both the Apriori algorithm and the FP-growth algorithm on this dataset are identical although the sequence of result is different in FP-growth.

Apriori Algorithm Result

```
apriori_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(other vegetables)	(whole milk)	0.058708	0.073679	0.002430	0.041389	0.561739	-0.001896	0.966315	-0.453207
1	(whole milk)	(other vegetables)	0.073679	0.058708	0.002430	0.032979	0.561739	-0.001896	0.973393	-0.457183
2	(rolls/buns)	(shopping bags)	0.068976	0.046402	0.002273	0.032955	0.710193	-0.000928	0.986094	-0.304735
3	(shopping bags)	(rolls/buns)	0.046402	0.068976	0.002273	0.048986	0.710193	-0.000928	0.978980	-0.299683
4	(rolls/buns)	(soda)	0.068976	0.066625	0.002351	0.034091	0.511684	-0.002244	0.966318	-0.506181
5	(soda)	(rolls/buns)	0.066625	0.068976	0.002351	0.035294	0.511684	-0.002244	0.965086	-0.505550
6	(rolls/buns)	(whole milk)	0.068976	0.073679	0.002587	0.037500	0.508963	-0.002496	0.962411	-0.508903
7	(whole milk)	(rolls/buns)	0.073679	0.068976	0.002587	0.035106	0.508963	-0.002496	0.964898	-0.510169
8	(shopping bags)	(soda)	0.046402	0.066625	0.002351	0.050676	0.760612	-0.000740	0.983199	-0.248146
9	(soda)	(shopping bags)	0.066625	0.046402	0.002351	0.035294	0.760612	-0.000740	0.988485	-0.252167
10	(whole milk)	(soda)	0.073679	0.066625	0.002508	0.034043	0.510959	-0.002401	0.966269	-0.508173
11	(soda)	(whole milk)	0.066625	0.073679	0.002508	0.037647	0.510959	-0.002401	0.962558	-0.506276
12	(yogurt)	(soda)	0.059962	0.066625	0.002587	0.043137	0.647465	-0.001408	0.975454	-0.366775
13	(soda)	(yogurt)	0.066625	0.059962	0.002587	0.038824	0.647465	-0.001408	0.978007	-0.368428

FP-growth Algorithm Result

```
fp_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(yogurt)	(soda)	0.059962	0.066625	0.002587	0.043137	0.647465	-0.001408	0.975454	-0.366775
1	(soda)	(yogurt)	0.066625	0.059962	0.002587	0.038824	0.647465	-0.001408	0.978007	-0.368428
2	(rolls/buns)	(whole milk)	0.068976	0.073679	0.002587	0.037500	0.508963	-0.002496	0.962411	-0.508903
3	(whole milk)	(rolls/buns)	0.073679	0.068976	0.002587	0.035106	0.508963	-0.002496	0.964898	-0.510169
4	(whole milk)	(soda)	0.073679	0.066625	0.002508	0.034043	0.510959	-0.002401	0.966269	-0.508173
5	(soda)	(whole milk)	0.066625	0.073679	0.002508	0.037647	0.510959	-0.002401	0.962558	-0.506276
6	(rolls/buns)	(soda)	0.068976	0.066625	0.002351	0.034091	0.511684	-0.002244	0.966318	-0.506181
7	(soda)	(rolls/buns)	0.066625	0.068976	0.002351	0.035294	0.511684	-0.002244	0.965086	-0.505550
8	(other vegetables)	(whole milk)	0.058708	0.073679	0.002430	0.041389	0.561739	-0.001896	0.966315	-0.453207
9	(whole milk)	(other vegetables)	0.073679	0.058708	0.002430	0.032979	0.561739	-0.001896	0.973393	-0.457183
10	(shopping bags)	(soda)	0.046402	0.066625	0.002351	0.050676	0.760612	-0.000740	0.983199	-0.248146
11	(soda)	(shopping bags)	0.066625	0.046402	0.002351	0.035294	0.760612	-0.000740	0.988485	-0.252167
12	(rolls/buns)	(shopping bags)	0.068976	0.046402	0.002273	0.032955	0.710193	-0.000928	0.986094	-0.304735
13	(shopping bags)	(rolls/buns)	0.046402	0.068976	0.002273	0.048986	0.710193	-0.000928	0.978980	-0.299683