Market Basket Analysis

January 7, 2021

1 BDA GROUP 13 PROJECT

No	Name	Matric Number	Task Distribution
1.	Fatimah binti Mohd Nizam	17218825	Codes & Objectives
2.	Xu Yizheng	17198425/2	Codes & Methodology
3.	Sharvind Gopal	S2018303	Codes & Introduction
4.	Lu Xian Ding	17096993	Codes & Discussion
5.	Nurfarhana binti Omar	17198278	Codes & Result

1.0.1 PART 1

1.1 Market Basket Analysis INTRODUCTION

Frequent Itemset

is a method for Market Basket Analysis. It involves sets of items with a defined minimum frequency. The set of items that equals or exceeds the minimum frequency is considered as a frequent itemset. It is utilized to find the frequently bought together items in the shopping behaviors of customers in a supermarket or in an online shopping platform.

Association Rule

mining has two essential phases: the first phase is to figure out the high frequency from the collected data and the second stage is to calculate the confidence value from the first stage result (Foxiao Zhan, 2019).

The latter stage is executed to identify all Large frequency sets items from the original data set. The minimum support threshold is identified to filter the itemsets that fulfilled the frequency requirement.

Support

is a measurement on how common or popular the items appear in the original dataset. The second stage is to generate involves confidence in its association rule mining process.

Confidence

is defined on how likely an item is purchased will affect the possibility for another item to be purchased. For example, confidence is a measurement of the proportion of transactions of item X with item Y appearing together. The drawback of confidence is that it only takes the item X's popularity instead of considering both. This could mislead the representation of the association importance.

Lift

There is another measurement called **lift** that controls the popularity of item Y while measuring how likely item Y will be purchased when item X is purchased. It is the ratio of the observed support to the expected if the two rules were independent from each other.

A lift value that is greater than 1 means that item Y is likely to be purchase with item X while a lift value which is less than 1 indicates the vice versa. Besides that, conviction is a measurement that implies on the strength of the rule from statistical independence (Dinesh J. Prajapati, 2017). It is defined:

Conviction

compares the probability that X appears without Y when they are dependent with each other with the actual frequency of the appearance X without Y. Conviction will have a value of 1 when the Items X and Y are completely unrelated. It is a directed measure because it also takes the value of appearance of X without Y into consideration.

OBJECTIVE OF THE REPORT

1. To perform Frequent Itemsets and Association Rules Mining towards a dataset that contains a list of items to analyze which items are frequently bought together.
2. To analyze the pattern and insight from the analysis based on the support, confidence, lift and conviction results.

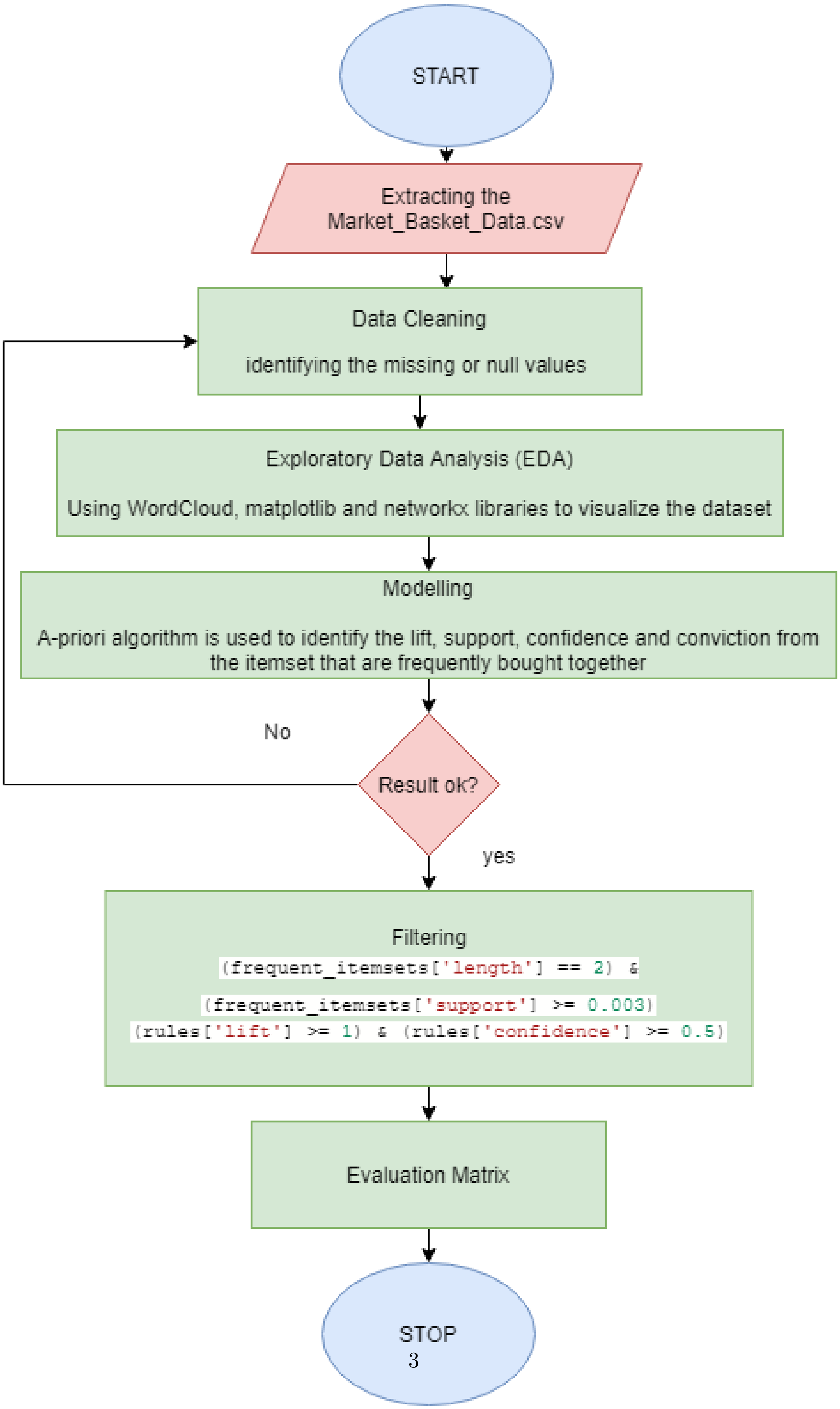
METHODOLOGY

Frequent itemset and Association rules mining are used to evaluate which items are frequently bought together in the data that has been collected. The Market_Basket_Data.csv is used in the study to study the implementation of frequent itemset and association rule. The data is initially cleaned to identify the amount of missing or null values. The missing or null values are ignored because the list of missing items does not affect the result of the methods used. Exploratory Data Analysis (EDA) process involves in visualizing the top popular items by using WordCloud. The frequency of the most popular items are plotted by using matplotlib. Networkx library is used to discover the top item choices in the dataset.

The modelling process is involved with implementing the A-priori algorithm to determine which antecedent items and consequent items are frequently bought together. The support, confidence, lift and conviction results are identified to understand deeper regarding the itemset purchase.

[1]:
```
from IPython.display import Image
Image(filename='BDA.png')
```

[1]:


START
Extracting the Market_Basket_Data.csv
Data Cleaning
identifying the missing or null values
Exploratory Data Analysis (EDA)
Using WordCloud, matplotlib and networkx libraries to visualize the dataset
Modelling
A-priori algorithm is used to identify the lift, support, confidence and conviction from the itemset that are frequently bought together
No
Result ok?
yes
Filtering
(frequent_itemsets['length'] == 2) &
(frequent_itemsets['support'] >= 0.003)
(rules['lift'] >= 1) & (rules['confidence'] >= 0.5)
Evaluation Matrix
STOP

INTRODUCTION TO DATASET

This dataset contains 7501 observations which showing groceries items and what usually being bought together with the items. Each observation contains different volume of columns but the most highest column is 20.Any null value is replaced with NaN.

1. Data Extraction

[2]:
```
!pip install squarify
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting squarify
  Downloading squarify-0.4.3-py3-none-any.whl (4.3 kB)
Installing collected packages: squarify
Successfully installed squarify-0.4.3
WARNING: You are using pip version 20.2.4; however, version 20.3.3 is
available.
You should consider upgrading via the '/usr/local/bin/python3.7 -m pip install
--upgrade pip' command.
```

[3]:
```
# for basic operations
import numpy as np
import pandas as pd

# for visualizations
import matplotlib.pyplot as plt
import squarify
import seaborn as sns
plt.style.use('fivethirtyeight')

# for market basket analysis
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

[5]:
```
# Assign url of file: url
url = 'https://raw.githubusercontent.com/FanaOmar/Market-Basket/main/
↪Market_Basket_Data.csv'

# Read file into a DataFrame: df
df = pd.read_csv(url, header= None)

# Print the head of the DataFrame
df.head(5)
```

[5]:
```
              0          1            2                 3              4  \
0        shrimp    almonds      avocado    vegetables mix   green grapes
1       burgers  meatballs         eggs               NaN            NaN
2       chutney        NaN          NaN               NaN            NaN
3        turkey    avocado          NaN               NaN            NaN
4 mineral water       milk   energy bar  whole wheat rice      green tea

                 5     6                7              8             9  \
0 whole weat flour  yams  cottage cheese   energy drink  tomato juice
1              NaN   NaN             NaN            NaN           NaN
2              NaN   NaN             NaN            NaN           NaN
3              NaN   NaN             NaN            NaN           NaN
4              NaN   NaN             NaN            NaN           NaN

              10          11      12      13             14      15  \
0  low fat yogurt   green tea   honey   salad   mineral water  salmon
1            NaN         NaN     NaN     NaN            NaN     NaN
2            NaN         NaN     NaN     NaN            NaN     NaN
3            NaN         NaN     NaN     NaN            NaN     NaN
4            NaN         NaN     NaN     NaN            NaN     NaN

                  16                17        18         19
0  antioxydant juice   frozen smoothie   spinach  olive oil
1                NaN               NaN       NaN        NaN
2                NaN               NaN       NaN        NaN
3                NaN               NaN       NaN        NaN
4                NaN               NaN       NaN        NaN
```

2. Data Understanding

[6]:
```python
df.shape
```

[6]: (7501, 20)

There are 7501 observations in 20 features in the dataset.

[7]:
```python
# Print few rows from the bottom of the DataFrame
df.tail()
```

[7]:
```
             0                   1             2                3          4  \
7496    butter          light mayo   fresh bread              NaN        NaN
7497   burgers   frozen vegetables          eggs     french fries  magazines
7498   chicken                 NaN           NaN              NaN        NaN
7499  escalope           green tea           NaN              NaN        NaN
7500      eggs     frozen smoothie   yogurt cake   low fat yogurt        NaN

           5    6    7    8    9   10   11   12   13   14   15   16   17  \
```

```
7496        NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
7497  green tea  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
7498        NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
7499        NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
7500        NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN

       18   19
7496  NaN  NaN
7497  NaN  NaN
7498  NaN  NaN
7499  NaN  NaN
7500  NaN  NaN
```

[8]:
```python
# checking the random entries in the data
df.sample(10)
```

[8]:
```
                   0                  1                  2               3  \
718        chocolate      grated cheese      mineral water          salmon
3520       chocolate          spaghetti      mineral water            soup
6751       chocolate  frozen vegetables  whole wheat pasta   mineral water
5144   mineral water            chicken        blueberries     fresh bread
5069      fresh tuna  frozen vegetables     low fat yogurt             NaN
2444            cake       french fries                NaN             NaN
2021        tomatoes               eggs            chicken chocolate bread
4155      fresh tuna      mineral water               eggs             NaN
6195         burgers            almonds               eggs         chicken
2280     ground beef               milk               cake             NaN

                    4             5                  6                     7  \
718  whole wheat rice  burger sauce           escalope  mushroom cream sauce
3520         pancakes          eggs   hand protein bar                   NaN
6751        olive oil    energy bar            chicken            white wine
5144       white wine     magazines                NaN                   NaN
5069              NaN           NaN                NaN                   NaN
2444              NaN           NaN                NaN                   NaN
2021   low fat yogurt           NaN                NaN                   NaN
4155              NaN           NaN                NaN                   NaN
6195       light mayo           NaN                NaN                   NaN
2280              NaN           NaN                NaN                   NaN

                  8    9   10   11   12   13   14   15   16   17   18   19
718  low fat yogurt  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
3520            NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6751            NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
5144            NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
5069            NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2444            NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
```

```
2021              NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4155              NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
6195              NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2280              NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
```

[9]:

```
df.describe()
```

[9]:

```
                    0              1              2              3          4  \
count            7501           5747           4389           3345       2529
unique            115            117            115            114        110
top     mineral water  mineral water  mineral water  mineral water  green tea
freq              577            484            375            201        153

                   5          6          7          8          9  \
count           1864       1369        981        654        395
unique           106        102         98         88         80
top     french fries  green tea  green tea  green tea  green tea
freq             107         96         67         57         31

                    10         11         12         13         14  \
count              256        154         87         47         25
unique              66         50         43         28         19
top     low fat yogurt  green tea  green tea  green tea  magazines
freq                22         15          8          4          3

                 15               16           17       18         19
count             8                4            4        3          1
unique            8                3            3        3          1
top     protein bar  frozen smoothie  protein bar  cereals  olive oil
freq              1                2            2        1          1
```

[10]:

```
#To check null values
df.isnull().sum()
```

[10]:

```
0        0
1     1754
2     3112
3     4156
4     4972
5     5637
6     6132
7     6520
8     6847
9     7106
10    7245
11    7347
12    7414
```

```
13    7454
14    7476
15    7493
16    7497
17    7497
18    7498
19    7500
dtype: int64
```

The NAN values are ignored due to the large amount of items in several columns are not identified. This will not affect our analysis process using A-priori algorithm.

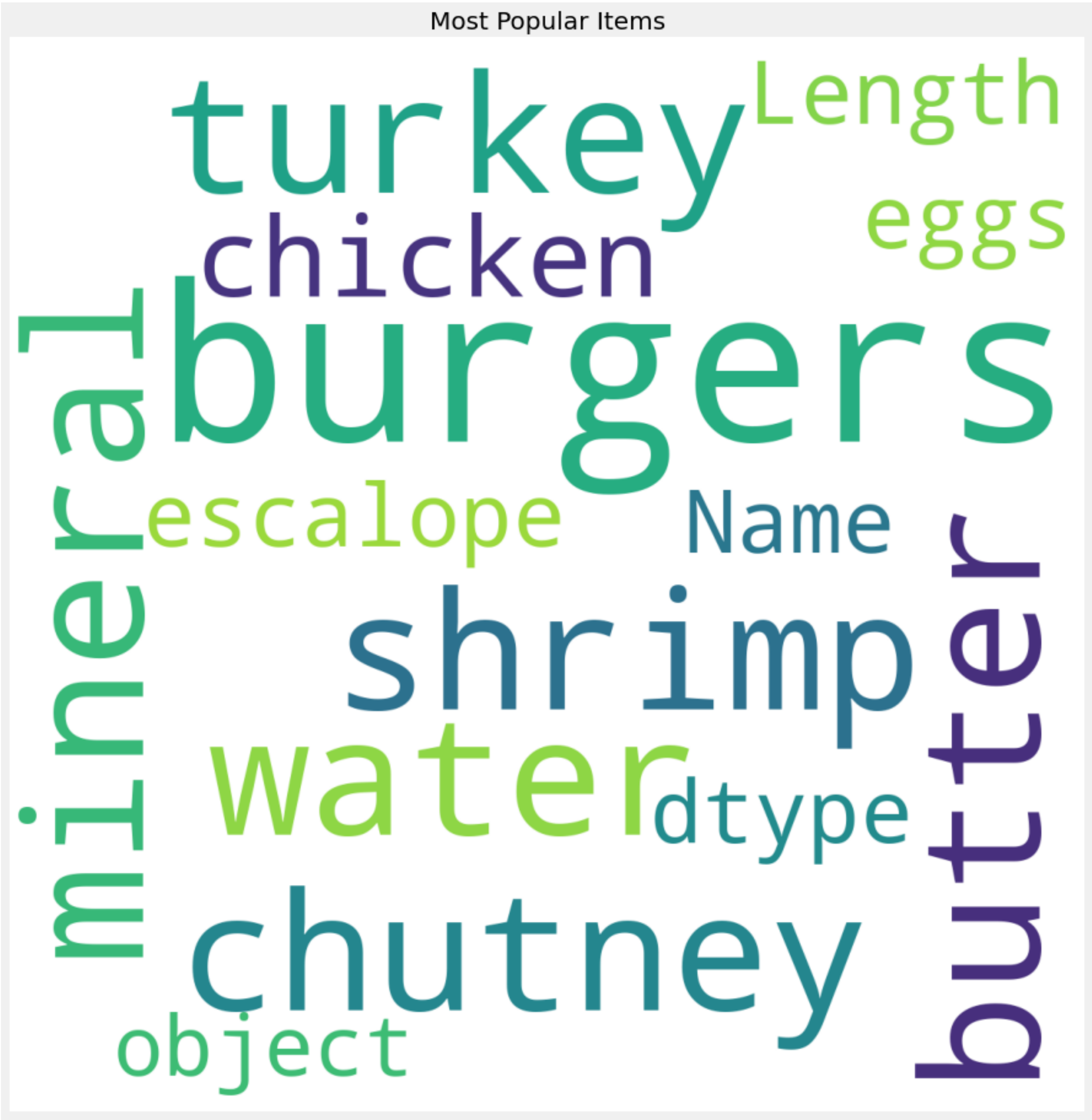
3. Exploratory Data Analysis (EDA)

[11]:
```
!pip install wordcloud
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: wordcloud in
/home/fatimahnizam/.local/lib/python3.7/site-packages (1.8.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/site-packages
(from wordcloud) (7.2.0)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.7/site-
packages (from wordcloud) (1.18.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/site-
packages (from wordcloud) (3.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/usr/local/lib/python3.7/site-packages (from matplotlib->wordcloud) (2.4.7)
Requirement already satisfied: certifi>=2020.06.20 in
/usr/local/lib/python3.7/site-packages (from matplotlib->wordcloud) (2020.6.20)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/site-packages (from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/site-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/site-
packages (from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/site-
packages (from python-dateutil>=2.1->matplotlib->wordcloud) (1.15.0)
WARNING: You are using pip version 20.2.4; however, version 20.3.3 is
available.
You should consider upgrading via the '/usr/local/bin/python3.7 -m pip install
--upgrade pip' command.
```

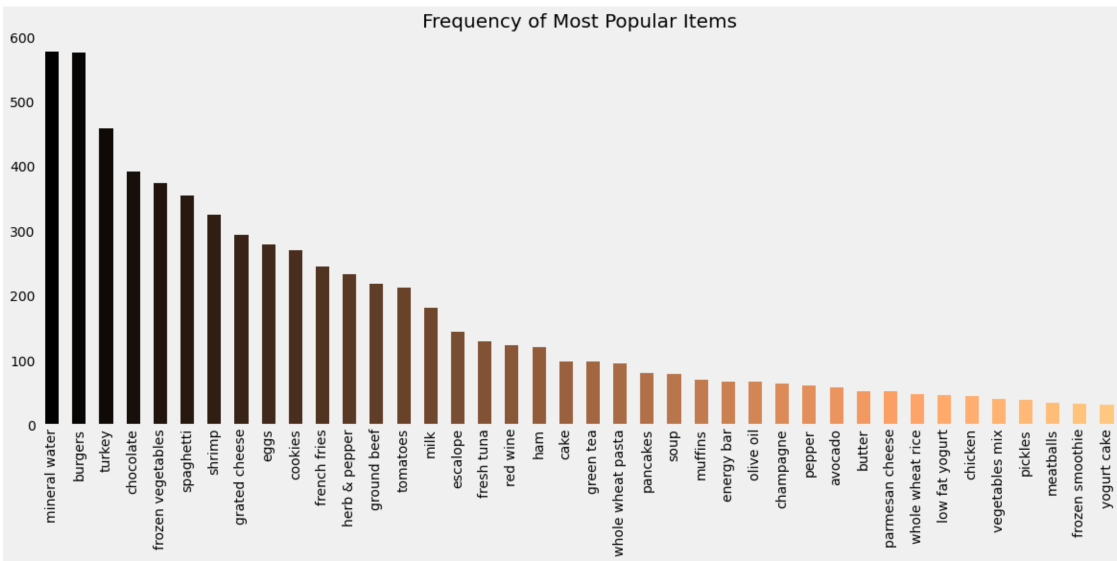
[12]:
```
# Data Visualization
# Create Wordcloud to visualize the most popular items in the DataFrame
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from wordcloud import WordCloud

plt.rcParams['figure.figsize'] = (15, 15)
wordcloud = WordCloud(background_color = 'white', width = 1200,  height = 1200,␣
 ↪max_words = 121).generate(str(df[0]))
plt.imshow(wordcloud)
plt.axis('off')
plt.title('Most Popular Items',fontsize = 20)
plt.show()
```



The most popular items in the data set are burgers, shrimp, turkey, mineral and many more. These are the items that the customers usually purchased.

[13]:
```
# Create a graph to visualize the frequency of most popular items

plt.rcParams['figure.figsize'] = (18, 7)
color = plt.cm.copper(np.linspace(0, 1, 40))
df[0].value_counts().head(40).plot.bar(color = color)
plt.title('Frequency of Most Popular Items', fontsize = 20)
plt.xticks(rotation = 90 )
plt.grid()
plt.show()
```



Mineral water is the highest item purchased in the store.

[15]:
```
df['food'] = 'Food'
food = df.truncate(before = -1, after = 15)


import networkx as nx

food = nx.from_pandas_edgelist(food, source = 'food', target = 0, edge_attr =␣
 ↪True)
```

[16]:
```
# To visualize Top 15 First Choices
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (20, 20)
pos = nx.spring_layout(food)
```

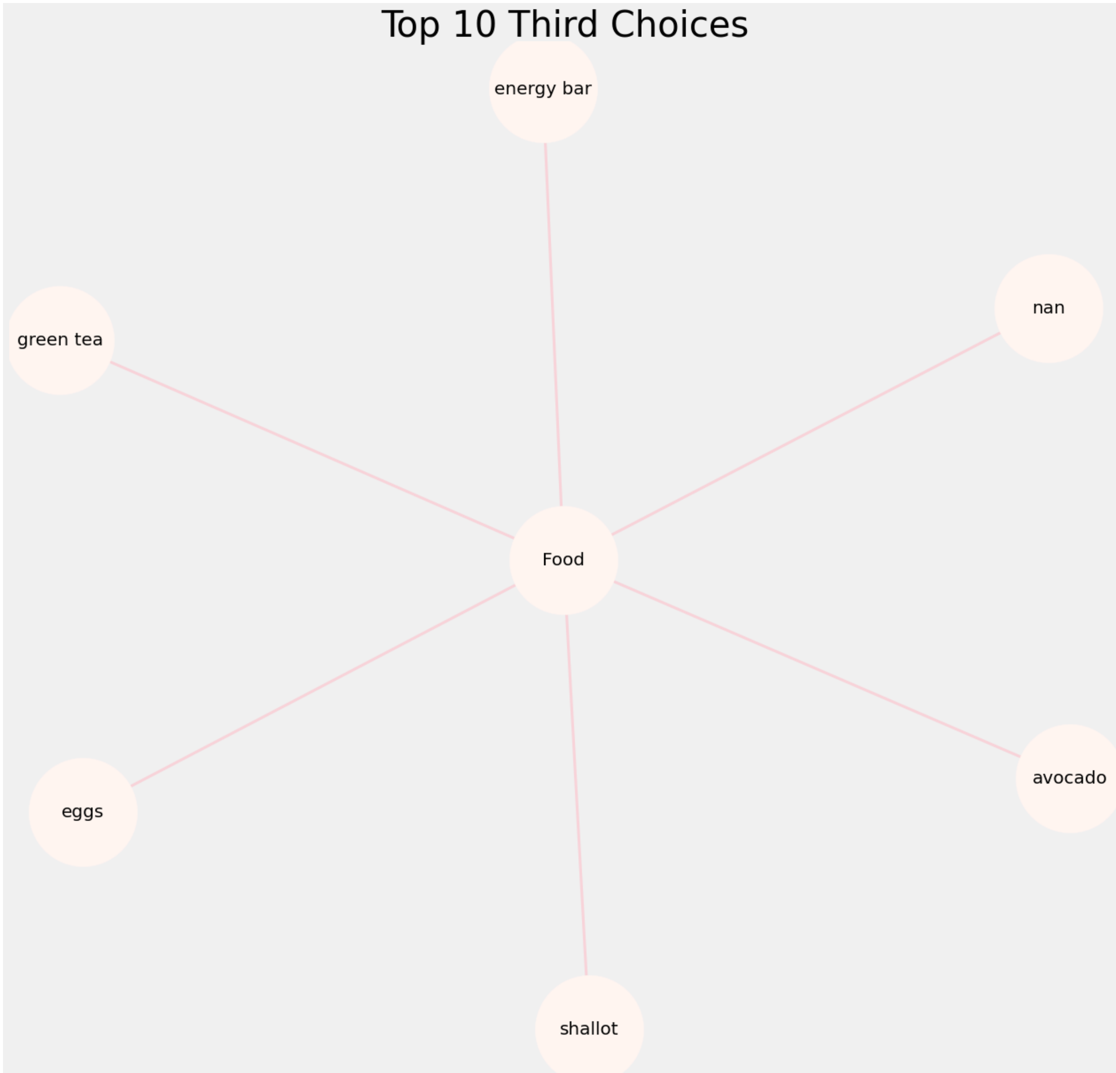
```
color = plt.cm.Wistia(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(food, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(food, pos, width = 3, alpha = 0.6, edge_color = 'black')
nx.draw_networkx_labels(food, pos, font_size = 20, font_family = 'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 15 First Choices', fontsize = 40)
plt.show()
```

[17]:
```
df['secondchoice'] = 'Second Choice'
secondchoice = df.truncate(before = -1, after = 15)
secondchoice = nx.from_pandas_edgelist(secondchoice, source = 'food', target =␣
 ↪1, edge_attr = True)
```

[18]:
```python
# To visualize Top 15 Second Choices
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (20, 20)
pos = nx.spring_layout(secondchoice)
color = plt.cm.Blues(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(secondchoice, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(secondchoice, pos, width = 3, alpha = 0.6, edge_color =␣
↪'brown')
nx.draw_networkx_labels(secondchoice, pos, font_size = 20, font_family =␣
↪'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 15 Second Choices', fontsize = 40)
plt.show()
```

[19]:
```
df['thirdchoice'] = 'Third Choice'
secondchoice = df.truncate(before = -1, after = 10)
secondchoice = nx.from_pandas_edgelist(secondchoice, source = 'food', target =␣
↪2, edge_attr = True)
```

[20]:
```
# To visualize Top 10 Third Choices

import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (20, 20)
pos = nx.spring_layout(secondchoice)
color = plt.cm.Reds(np.linspace(0, 15, 1))
nx.draw_networkx_nodes(secondchoice, pos, node_size = 15000, node_color = color)
nx.draw_networkx_edges(secondchoice, pos, width = 3, alpha = 0.6, edge_color =␣
↪'pink')
nx.draw_networkx_labels(secondchoice, pos, font_size = 20, font_family =␣
↪'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 10 Third Choices', fontsize = 40)
plt.show()
```



[21]:
```
# making each customers shopping items an identical list
trans = []
for i in range(0, 7501):
    trans.append([str(df.values[i,j]) for j in range(0, 20)])

# converting it into an numpy array
trans = np.array(trans)

# checking the shape of the array
print(trans.shape)
```

```
(7501, 20)
```

[22]:
```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
data = te.fit_transform(trans)
data = pd.DataFrame(data, columns = te.columns_)

# getting the shape of the data
data.shape
```

[22]: (7501, 121)

[23]:
```python
import warnings
warnings.filterwarnings('ignore')

# getting correlations for 121 items would be messy
# so let's reduce the items from 121 to 50

data = data.loc[:, ['mineral water', 'burgers', 'turkey', 'chocolate', 'frozen␣
↪vegetables', 'spaghetti',
                    'shrimp', 'grated cheese', 'eggs', 'cookies', 'french␣
↪fries', 'herb & pepper', 'ground beef',
                    'tomatoes', 'milk', 'escalope', 'fresh tuna', 'red wine',␣
↪'ham', 'cake', 'green tea',
                    'whole wheat pasta', 'pancakes', 'soup', 'muffins', 'energy␣
↪bar', 'olive oil', 'champagne',
                    'avocado', 'pepper', 'butter', 'parmesan cheese', 'whole␣
↪wheat rice', 'low fat yogurt',
                   'chicken', 'vegetables mix', 'pickles', 'meatballs', 'frozen␣
↪smoothie', 'yogurt cake']]

# checking the shape
data.shape
```

[23]: (7501, 40)

[24]:
```python
# let's check the columns
data.columns
```

[24]:
```
Index(['mineral water', 'burgers', 'turkey', 'chocolate', 'frozen vegetables',
       'spaghetti', 'shrimp', 'grated cheese', 'eggs', 'cookies',
       'french fries', 'herb & pepper', 'ground beef', 'tomatoes', 'milk',
       'escalope', 'fresh tuna', 'red wine', 'ham', 'cake', 'green tea',
       'whole wheat pasta', 'pancakes', 'soup', 'muffins', 'energy bar',
       'olive oil', 'champagne', 'avocado', 'pepper', 'butter',
       'parmesan cheese', 'whole wheat rice', 'low fat yogurt', 'chicken',
       'vegetables mix', 'pickles', 'meatballs', 'frozen smoothie',
```

```
 'yogurt cake'],
dtype='object')
```

[25]:
```
# getting the head of the data
data.head()
```

[25]:
```
   mineral water  burgers  turkey  chocolate  frozen vegetables  spaghetti  \
0           True    False   False      False              False      False
1          False     True   False      False              False      False
2          False    False   False      False              False      False
3          False    False    True      False              False      False
4           True    False   False      False              False      False

   shrimp  grated cheese   eggs  cookies  …  butter  parmesan cheese  \
0    True          False  False    False  …   False            False
1   False          False   True    False  …   False            False
2   False          False  False    False  …   False            False
3   False          False  False    False  …   False            False
4   False          False  False    False  …   False            False

   whole wheat rice  low fat yogurt  chicken  vegetables mix  pickles  \
0             False            True    False            True    False
1             False           False    False           False    False
2             False           False    False           False    False
3             False           False    False           False    False
4              True           False    False           False    False

   meatballs  frozen smoothie  yogurt cake
0      False             True        False
1       True            False        False
2      False            False        False
3      False            False        False
4      False            False        False

[5 rows x 40 columns]
```

4. Apriori Algorithm This algorrithm is used to find the frequent itemsets and association rules.

Condition I. Value Set as, Support : 0.003 Length : 2

[26]:
```
from mlxtend.frequent_patterns import apriori

#Now, let us return the items and itemsets with 3% support:
apriori(data, min_support = 0.003, use_colnames = True)
```

```
[26]:       support                                        itemsets
     0    0.238368                                 (mineral water)
     1    0.087188                                       (burgers)
     2    0.062525                                        (turkey)
     3    0.163845                                     (chocolate)
     4    0.095321                             (frozen vegetables)
     ..        …                                             …
     995  0.003333           (mineral water, eggs, ground beef, milk)
     996  0.003066  (spaghetti, chocolate, frozen vegetables, grou…
     997  0.003466    (chocolate, frozen vegetables, spaghetti, milk)
     998  0.003066          (chocolate, eggs, spaghetti, ground beef)
     999  0.003066  (spaghetti, frozen vegetables, ground beef, milk)

     [1000 rows x 2 columns]
```

```
[27]: frequent_itemsets = apriori(data, min_support = 0.003, use_colnames=True)
      frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:␣
       ↪len(x))
      frequent_itemsets
```

```
[27]:       support                                        itemsets  length
     0    0.238368                                 (mineral water)       1
     1    0.087188                                       (burgers)       1
     2    0.062525                                        (turkey)       1
     3    0.163845                                     (chocolate)       1
     4    0.095321                             (frozen vegetables)       1
     ..        …                                             …      …
     995  0.003333           (mineral water, eggs, ground beef, milk)       4
     996  0.003066  (spaghetti, chocolate, frozen vegetables, grou…       4
     997  0.003466    (chocolate, frozen vegetables, spaghetti, milk)       4
     998  0.003066          (chocolate, eggs, spaghetti, ground beef)       4
     999  0.003066  (spaghetti, frozen vegetables, ground beef, milk)       4

     [1000 rows x 3 columns]
```

```
[28]: # getting the item sets with length = 2 and support more than 3%

      frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                         (frequent_itemsets['support'] >= 0.003) ]
```

```
[28]:      support                           itemsets  length
     40  0.024397             (mineral water, burgers)       2
     41  0.019197              (mineral water, turkey)       2
     42  0.052660           (mineral water, chocolate)       2
     43  0.035729  (mineral water, frozen vegetables)       2
     44  0.059725           (mineral water, spaghetti)       2
     ..       …                                 …      …
```

```
495  0.004399           (chicken, low fat yogurt)       2
496  0.007332   (frozen smoothie, low fat yogurt)       2
497  0.003200                (chicken, meatballs)       2
498  0.006666          (chicken, frozen smoothie)       2
499  0.003333   (frozen smoothie, vegetables mix)       2

[460 rows x 3 columns]
```

Condition II. Value Set as, Support : 0.004 Length : 2

[29]:
```python
from mlxtend.frequent_patterns import apriori

#Now, let us return the items and itemsets with at least 4% support:
apriori(data, min_support = 0.004, use_colnames = True)
```

[29]:
```
      support                                         itemsets
0    0.238368                                  (mineral water)
1    0.087188                                        (burgers)
2    0.062525                                         (turkey)
3    0.163845                                      (chocolate)
4    0.095321                              (frozen vegetables)
..        …                                              …
670  0.004133           (mineral water, eggs, milk, chocolate)
671  0.004399  (mineral water, spaghetti, frozen vegetables, …
672  0.004533  (mineral water, frozen vegetables, spaghetti, …
673  0.004399           (mineral water, eggs, spaghetti, milk)
674  0.004399     (mineral water, spaghetti, ground beef, milk)

[675 rows x 2 columns]
```

[30]:
```python
frequent_itemsets = apriori(data, min_support = 0.004, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:␣
 ↪len(x))
frequent_itemsets
```

[30]:
```
      support                                         itemsets  length
0    0.238368                                  (mineral water)       1
1    0.087188                                        (burgers)       1
2    0.062525                                         (turkey)       1
3    0.163845                                      (chocolate)       1
4    0.095321                              (frozen vegetables)       1
..        …                                              …     …
670  0.004133           (mineral water, eggs, milk, chocolate)       4
671  0.004399  (mineral water, spaghetti, frozen vegetables, …       4
672  0.004533  (mineral water, frozen vegetables, spaghetti, …       4
673  0.004399           (mineral water, eggs, spaghetti, milk)       4
674  0.004399     (mineral water, spaghetti, ground beef, milk)       4
```

```
[675 rows x 3 columns]
```

[32]:
```
# getting the item sets with length = 2 and support more than 4%

frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                   (frequent_itemsets['support'] >= 0.004) ]
```

[32]:
```
      support                            itemsets  length
40   0.024397            (mineral water, burgers)       2
41   0.019197             (mineral water, turkey)       2
42   0.052660          (mineral water, chocolate)       2
43   0.035729  (mineral water, frozen vegetables)       2
44   0.059725          (mineral water, spaghetti)       2
..        ...                                 ...     ...
416  0.005466          (whole wheat rice, chicken)      2
417  0.005999 (whole wheat rice, frozen smoothie)       2
418  0.004399            (chicken, low fat yogurt)      2
419  0.007332    (frozen smoothie, low fat yogurt)      2
420  0.006666          (chicken, frozen smoothie)       2

[381 rows x 3 columns]
```

Condition III. Value Set as, Support : 0.004 Length : 4

[34]:
```
from mlxtend.frequent_patterns import apriori

#Now, let us return the items and itemsets with at least 4% support:
apriori(data, min_support = 0.004, use_colnames = True)
```

[34]:
```
      support                                           itemsets
0    0.238368                                    (mineral water)
1    0.087188                                          (burgers)
2    0.062525                                           (turkey)
3    0.163845                                        (chocolate)
4    0.095321                                (frozen vegetables)
..        ...                                                ...
670  0.004133            (mineral water, eggs, milk, chocolate)
671  0.004399  (mineral water, spaghetti, frozen vegetables, …
672  0.004533  (mineral water, frozen vegetables, spaghetti, …
673  0.004399             (mineral water, eggs, spaghetti, milk)
674  0.004399      (mineral water, spaghetti, ground beef, milk)

[675 rows x 2 columns]
```

[35]:
```
frequent_itemsets = apriori(data, min_support = 0.004, use_colnames=True)
```

```
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:␣
↪len(x))
frequent_itemsets
```

[35]:

```
        support                                        itemsets  length
0    0.238368                                 (mineral water)       1
1    0.087188                                       (burgers)       1
2    0.062525                                        (turkey)       1
3    0.163845                                     (chocolate)       1
4    0.095321                             (frozen vegetables)       1
..        …                                             …     …
670  0.004133            (mineral water, eggs, milk, chocolate)       4
671  0.004399  (mineral water, spaghetti, frozen vegetables, …       4
672  0.004533  (mineral water, frozen vegetables, spaghetti, …       4
673  0.004399            (mineral water, eggs, spaghetti, milk)       4
674  0.004399      (mineral water, spaghetti, ground beef, milk)       4

[675 rows x 3 columns]
```

[36]:

```
# getting the item sets with length = 4 and support more than 4%

frequent_itemsets[ (frequent_itemsets['length'] == 4) &
                   (frequent_itemsets['support'] >= 0.004) ]
```

[36]:

```
        support                                        itemsets  length
667  0.004133  (mineral water, chocolate, frozen vegetables, …       4
668  0.004533        (mineral water, eggs, spaghetti, chocolate)       4
669  0.004933        (mineral water, chocolate, spaghetti, milk)       4
670  0.004133            (mineral water, eggs, milk, chocolate)       4
671  0.004399  (mineral water, spaghetti, frozen vegetables, …       4
672  0.004533  (mineral water, frozen vegetables, spaghetti, …       4
673  0.004399            (mineral water, eggs, spaghetti, milk)       4
674  0.004399      (mineral water, spaghetti, ground beef, milk)       4
```

[37]:

```
# To find the value of the support, confidence, lift and conviction
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()
```

[37]:

```
       antecedents      consequents  antecedent support  consequent support  \
0  (mineral water)        (burgers)            0.238368            0.087188
1        (burgers)  (mineral water)            0.087188            0.238368
2  (mineral water)         (turkey)            0.238368            0.062525
3         (turkey)  (mineral water)            0.062525            0.238368
4  (mineral water)      (chocolate)            0.238368            0.163845

    support  confidence      lift  leverage  conviction
0  0.024397    0.102349  1.173883  0.003614    1.016889
```

```
1  0.024397    0.279817  1.173883  0.003614    1.057552
2  0.019197    0.080537  1.288075  0.004293    1.019590
3  0.019197    0.307036  1.288075  0.004293    1.099093
4  0.052660    0.220917  1.348332  0.013604    1.073256
```

[38]:
```python
# To filter the itemsets that has lift value equals or more than 1 and value of
# confidence equals or more than 0.5
rules[ (rules['lift'] >= 1) &
       (rules['confidence'] >= 0.5) ]
```

[38]:
```
                                       antecedents      consequents  \
714                    (frozen vegetables, turkey)  (mineral water)
732                                (turkey, milk)  (mineral water)
806                             (chocolate, soup)  (mineral water)
812                        (chocolate, olive oil)  (mineral water)
824                          (chicken, chocolate)  (mineral water)
860               (frozen vegetables, ground beef)  (mineral water)
894                      (soup, frozen vegetables)  (mineral water)
900                 (olive oil, frozen vegetables)  (mineral water)
988                             (soup, spaghetti)  (mineral water)
1048                          (shrimp, olive oil)  (mineral water)
1074                          (eggs, ground beef)  (mineral water)
1110                                 (eggs, soup)  (mineral water)
1190                          (ground beef, milk)  (mineral water)
1200                      (pancakes, ground beef)  (mineral water)
1206                          (soup, ground beef)  (mineral water)
1218              (ground beef, low fat yogurt)  (mineral water)
1236                        (olive oil, tomatoes)  (mineral water)
1258                                 (soup, milk)  (mineral water)
1264                            (olive oil, milk)  (mineral water)
1318                             (soup, pancakes)  (mineral water)
1330                  (whole wheat rice, pancakes)  (mineral water)
1342                          (chicken, pancakes)  (mineral water)
1348                            (soup, olive oil)  (mineral water)
1714              (frozen vegetables, ground beef)      (spaghetti)
1742                (olive oil, frozen vegetables)      (spaghetti)
1803                        (shrimp, ground beef)      (spaghetti)
1939                       (chicken, ground beef)      (spaghetti)
1951                        (tomatoes, olive oil)      (spaghetti)
2115     (chocolate, frozen vegetables, spaghetti)  (mineral water)
2171   (frozen vegetables, spaghetti, ground beef)  (mineral water)
2185          (frozen vegetables, spaghetti, milk)  (mineral water)

      antecedent support  consequent support   support  confidence      lift  \
714             0.008799            0.238368  0.004399    0.500000  2.097595
732             0.011332            0.238368  0.006133    0.541176  2.270338
806             0.010132            0.238368  0.005599    0.552632  2.318395
```

812	0.016398	0.238368	0.008266	0.504065	2.114649
824	0.014665	0.238368	0.007599	0.518182	2.173871
860	0.016931	0.238368	0.009199	0.543307	2.279277
894	0.007999	0.238368	0.005066	0.633333	2.656954
900	0.011332	0.238368	0.006532	0.576471	2.418404
988	0.014265	0.238368	0.007466	0.523364	2.195614
1048	0.008132	0.238368	0.004533	0.557377	2.338303
1074	0.019997	0.238368	0.010132	0.506667	2.125563
1110	0.009065	0.238368	0.004933	0.544118	2.282677
1190	0.021997	0.238368	0.011065	0.503030	2.110308
1200	0.014531	0.238368	0.007466	0.513761	2.155327
1206	0.009732	0.238368	0.005066	0.520548	2.183798
1218	0.009599	0.238368	0.004799	0.500000	2.097595
1236	0.007199	0.238368	0.004133	0.574074	2.408350
1258	0.015198	0.238368	0.008532	0.561404	2.355194
1264	0.017064	0.238368	0.008532	0.500000	2.097595
1318	0.006799	0.238368	0.004266	0.627451	2.632276
1330	0.006932	0.238368	0.004133	0.596154	2.500979
1342	0.009065	0.238368	0.004799	0.529412	2.220983
1348	0.008932	0.238368	0.005199	0.582090	2.441976
1714	0.016931	0.174110	0.008666	0.511811	2.939582
1742	0.011332	0.174110	0.005733	0.505882	2.905531
1803	0.011465	0.174110	0.005999	0.523256	3.005315
1939	0.009465	0.174110	0.004799	0.507042	2.912193
1951	0.007199	0.174110	0.004399	0.611111	3.509912
2115	0.007866	0.238368	0.004133	0.525424	2.204252
2171	0.008666	0.238368	0.004399	0.507692	2.129866
2185	0.008266	0.238368	0.004533	0.548387	2.300588

	leverage	conviction
714	0.002302	1.523264
732	0.003431	1.659967
806	0.003184	1.702471
812	0.004357	1.535749
824	0.004103	1.580745
860	0.005163	1.667711
894	0.003159	2.077178
900	0.003831	1.798297
988	0.004065	1.597933
1048	0.002594	1.720724
1074	0.005365	1.543848
1110	0.002772	1.670676
1190	0.005822	1.532552
1200	0.004002	1.566375
1206	0.002746	1.588546
1218	0.002511	1.523264
1236	0.002417	1.788179

```
1258  0.004909    1.736520
1264  0.004465    1.523264
1318  0.002645    2.044380
1330  0.002480    1.885945
1342  0.002638    1.618468
1348  0.003070    1.822476
1714  0.005718    1.691742
1742  0.003760    1.671444
1803  0.004003    1.732354
1939  0.003151    1.675377
1951  0.003146    2.123717
2115  0.002258    1.604867
2171  0.002334    1.547065
2185  0.002562    1.686470
```

RESULT

The result from the a-priori algorithm is shown below.

[42]:
```python
from IPython.display import Image
Image(filename='B1.jpeg')
```

[42]:

Condition/Items	Support	Length	Findings
Condition I	0.003	2	460 association rules found
Condition II	0.004	2	381 association rules found
Condition III	0.004	4	8 association rules found

Table I

From the table above, the highest association rules found is when Support is set to 0.003 and length set to 2. The lowest association rules found when Support is set to 0.004 and length set to 4.

[43]:
```python
from IPython.display import Image
Image(filename='B2.jpeg')
```

[43]:

	support	itemsets	length
40	0.024397	(mineral water, burgers)	2
41	0.019197	(mineral water, turkey)	2
42	0.052660	(mineral water, chocolate)	2
43	0.035729	(mineral water, frozen vegetables)	2
44	0.059725	(mineral water, spaghetti)	2
...	...	...	...
495	0.004399	(chicken, low fat yogurt)	2
496	0.007332	(frozen smoothie, low fat yogurt)	2
497	0.003200	(chicken, meatballs)	2
498	0.006666	(frozen smoothie, chicken)	2
499	0.003333	(frozen smoothie, vegetables mix)	2

Figure I

Refer to the Figure I above, it is to find the support for each itemsets. The higher the support number, indicate that the itemsets occured frequently. For instance, itemsets (mineral water, spaghetti) with support value 0.059725 are occured more frequent than itemsets (mineral water, burgers) with support value 0.024397.

[44]:
```
from IPython.display import Image
Image(filename='B3.jpeg')
```

[44]:

	antecedents	consequents	ntecedent support	consequent support	support	confidence	lift	leverage	conviction
714	(frozen vegetables, turkey)	(mineral water)	0.008799	0.238368	0.004399	0.500000	2.097595	0.002302	1.523264
732	(milk, turkey)	(mineral water)	0.011332	0.238368	0.006133	0.541176	2.270338	0.003431	1.659967
806	(chocolate, soup)	(mineral water)	0.010132	0.238368	0.005599	0.552632	2.318395	0.003184	1.702471
812	(olive oil, chocolate)	(mineral water)	0.016398	0.238368	0.008266	0.504065	2.114649	0.004357	1.535749
824	(chicken, chocolate)	(mineral water)	0.014665	0.238368	0.007599	0.518182	2.173871	0.004103	1.580745
860	(ground beef, frozen vegetables)	(mineral water)	0.016931	0.238368	0.009199	0.543307	2.279277	0.005163	1.667711
894	(frozen vegetables, soup)	(mineral water)	0.007999	0.238368	0.005066	0.633333	2.656954	0.003159	2.077178
900	(frozen vegetables, olive oil)	(mineral water)	0.011332	0.238368	0.006532	0.576471	2.418404	0.003831	1.798297
988	(spaghetti, soup)	(mineral water)	0.014265	0.238368	0.007466	0.523364	2.195614	0.004065	1.597933
1048	(shrimp, olive oil)	(mineral water)	0.008132	0.238368	0.004533	0.557377	2.338303	0.002594	1.720724
1074	(eggs, ground beef)	(mineral water)	0.019997	0.238368	0.010132	0.506667	2.125563	0.005365	1.543848
1110	(eggs, soup)	(mineral water)	0.009065	0.238368	0.004933	0.544118	2.282677	0.002772	1.670676
1190	(milk, ground beef)	(mineral water)	0.021997	0.238368	0.011065	0.503030	2.110308	0.005822	1.532552
1200	(pancakes, ground beef)	(mineral water)	0.014531	0.238368	0.007466	0.513761	2.155327	0.004002	1.566375
1206	(ground beef, soup)	(mineral water)	0.009732	0.238368	0.005066	0.520548	2.183798	0.002746	1.588546
1218	(ground beef, low fat yogurt)	(mineral water)	0.009599	0.238368	0.004799	0.500000	2.097595	0.002511	1.523264
1236	(tomatoes, olive oil)	(mineral water)	0.007199	0.238368	0.004133	0.574074	2.408350	0.002417	1.788179
1258	(milk, soup)	(mineral water)	0.015198	0.238368	0.008532	0.561404	2.355194	0.004909	1.736520
1264	(milk, olive oil)	(mineral water)	0.017064	0.238368	0.008532	0.500000	2.097595	0.004465	1.523264
1318	(pancakes, soup)	(mineral water)	0.006799	0.238368	0.004266	0.627451	2.632276	0.002645	2.044380
1330	(pancakes, whole wheat rice)	(mineral water)	0.006932	0.238368	0.004133	0.596154	2.500979	0.002480	1.885945
1342	(pancakes, chicken)	(mineral water)	0.009065	0.238368	0.004799	0.529412	2.220983	0.002638	1.618468
1348	(olive oil, soup)	(mineral water)	0.008932	0.238368	0.005199	0.582090	2.441976	0.003070	1.822476
1714	(ground beef, frozen vegetables)	(spaghetti)	0.016931	0.174110	0.008666	0.511811	2.939582	0.005718	1.691742
1744	(frozen vegetables, olive oil)	(spaghetti)	0.011332	0.174110	0.005733	0.505882	2.905531	0.003760	1.671444
1803	(shrimp, ground beef)	(spaghetti)	0.011465	0.174110	0.005999	0.523256	3.005315	0.004003	1.732354
1938	(chicken, ground beef)	(spaghetti)	0.009465	0.174110	0.004799	0.507042	2.912193	0.003151	1.675377
1951	(tomatoes, olive oil)	(spaghetti)	0.007199	0.174110	0.004399	0.611111	3.509912	0.003146	2.123717

Figure II

Figure II above showing the key metrics of rules found. Those in box 1,or antecedents column is list of the itemsets you bought and the second box indicate with 2 or consequents, showing the item that you might buy if you buy itemsets in 1. Those columns in box 3, is a calculated metrices of the associated rules.

EVALUATION AND DISCUSSION

Now this is the part of figuring out what the data is telling us.

1. Lift

There are quite a few rules with a **high lift value** which means that it occurs more frequently than would be expected based on the given number of transaction and product combinations.

Lift values that are more than 1 could be indicative of a useful rule pattern (more likely to be bought together).

Lift value that is negative shows that there is a negative correlation.

Lift value that is positive shows that there is a positive correlation.

Lift value ratio that is equals to 1 shows that there the items are independent (no correlation).

2. Support

Support refers to how often a given rule appears in the dataset being mined. For example, we will consider the support regarding the purchase of (burgers, olive oil) with (mineral water). The value of support is 0.00333. This means that at least 33 times out of a total of 10,000 transactions that the itemset will occur.

3. Confidence

A confidence of 0.5 means that 50% of the cases where antecedent item and consequent item are purchased together. For example, there is a 50% chance that (turkey, frozen vegetables) and (mineral water) are bought together.

There is also chances where we can get high support but low confidence. This happens when a rule show a strong correlation in a dataset because it appears very often but may occur far less when applied.

4. Conviction

Compares the probability that antecedent item appears without consequent item if they were dependent. For example, the conviction rule for (burgers, olive oil) and (mineral water) was 1.589 means that the rule will be incorrect 59% more often if the association between (burgers, olive oil) and (mineral water) was purely random chance.

LIMITATION OF THE RESULTS

The limitation of the result is the difficulties to tell on which week or when does the frequently bought itemset occured based on the dataset. This is because the dates are not recorded in the dataset. It is hard to analyze and make a future prediction from the dataset given. It is essential for the owner to predict and plan their inventory cost to increase the profits and avoid any losses due to improper inventory planning.

CONCLUSION AND FUTURE WORK

From observations, mineral water is the most bought consequents item regardless of the antecedents itemsets category. Noticeable that people whom bought ground beef will highly bought the spaghetti together. It is also noticed that the lower value of support, lift and confidence is set, the more significant rules generated.

There are also many other complex alternatives that can be utilized to analyze the data such as regression, Neural Networks, clustering and many more. The challenges that most of data scientists faced when using the algorithms are, they can be difficult to tune, hard to be interpreted and in need of feature engineering to produce an excellent and accurate results. The algorithms require an intensive knowledge to implement them in the analysis.

Associate analysis is another option that requires a light mathematical concept, and it has a less complicated algorithm to explain to the non-technical people. It can be implemented in Python by using the MLxtend library. It is an unsupervised learning algorithm that searches for hidden patterns without completely relying on the data prep and feature engineering which eases the process of the data analysis.

This work can be further continued to analyze the peak seasons for the itemsets to be bought together. This will helps the owner to predict when and how much the inventory cost should be spent and thus helps to generate a better growing economy.

REREFENCES

1. https://www.kaggle.com/yugagrawal95/market-basket-analysis-apriori-in-python
2. https://pbpython.com/market-basket-analysis.html