Grocery Data Analysis and Association Rules

Created By:

Mukul Aggarwal

MSD23007

2

```
import seaborn as sns
import pandas as pd
from matplotlib import pyplot as plt
```

3

```
import plotly.express as px
import plotly.io as pio
```

4

```
# Load the table into a DataFrame
df = spark.table("Groceries_Transactions")
```

5

```
df.show()
```

```
|         2300|2015-09-19|           pip fruit|2015|    9| 19|        38|
|         1187|2015-12-12|    other vegetables|2015|   12| 12|        50|
|         3037|2015-02-01|          whole milk|2015|    2|  1|         5|
|         4941|2015-02-14|          rolls/buns|2015|    2| 14|         7|
|         4501|2015-05-08|    other vegetables|2015|    5|  8|        19|
|         3803|2015-12-23|          pot plants|2015|   12| 23|        52|
|         2762|2015-03-20|          whole milk|2015|    3| 20|        12|
|         4119|2015-02-12|      tropical fruit|2015|    2| 12|         7|
|         1340|2015-02-24|        citrus fruit|2015|    2| 24|         9|
|         2193|2015-04-14|                beef|2015|    4| 14|        16|
|         1997|2015-07-21|         frankfurter|2015|    7| 21|        30|
|         4546|2015-09-03|             chicken|2015|    9|  3|        36|
|         4736|2015-07-21|              butter|2015|    7| 21|        30|
|         1959|2015-03-30|fruit/vegetable j...|2015|    3| 30|        14|
|         1974|2015-05-03|packaged fruit/ve...|2015|    5|  3|        18|
|         2421|2015-09-02|           chocolate|2015|    9|  2|        36|
|         1513|2015-08-03|        specialty bar|2015|    8|  3|        32|
|         1905|2015-07-07|    other vegetables|2015|    7|  7|        28|
+-------------+----------+--------------------+----+-----+---+----------+
only showing top 20 rows
```

6

```
pandas_df = df.toPandas()
```

```
/databricks/spark/python/pyspark/sql/pandas/utils.py:51: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

/databricks/spark/python/pyspark/sql/pandas/utils.py:85: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

/databricks/spark/python/pyspark/sql/pandas/utils.py:85: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.

/databricks/spark/python/pyspark/sql/pandas/conversion.py:161: DeprecationWarning:
```

```
/databricks/spark/python/pyspark/sql/pandas/conversion.py:161: DeprecationWarning:

distutils Version classes are deprecated. Use packaging.version instead.
```

7

```
pandas_df.head()
```

	Member_number	Date	itemDescription	Year	Month	Day	WeekOfYear
0	1808	2015-07-21	tropical fruit	2015	7	21	30
1	2552	2015-01-05	whole milk	2015	1	5	2
2	2300	2015-09-19	pip fruit	2015	9	19	38
3	1187	2015-12-12	other vegetables	2015	12	12	50
4	3037	2015-02-01	whole milk	2015	2	1	5

Count of Unique Items

9

```
pandas_df['itemDescription'].unique().size
```

```
167
```

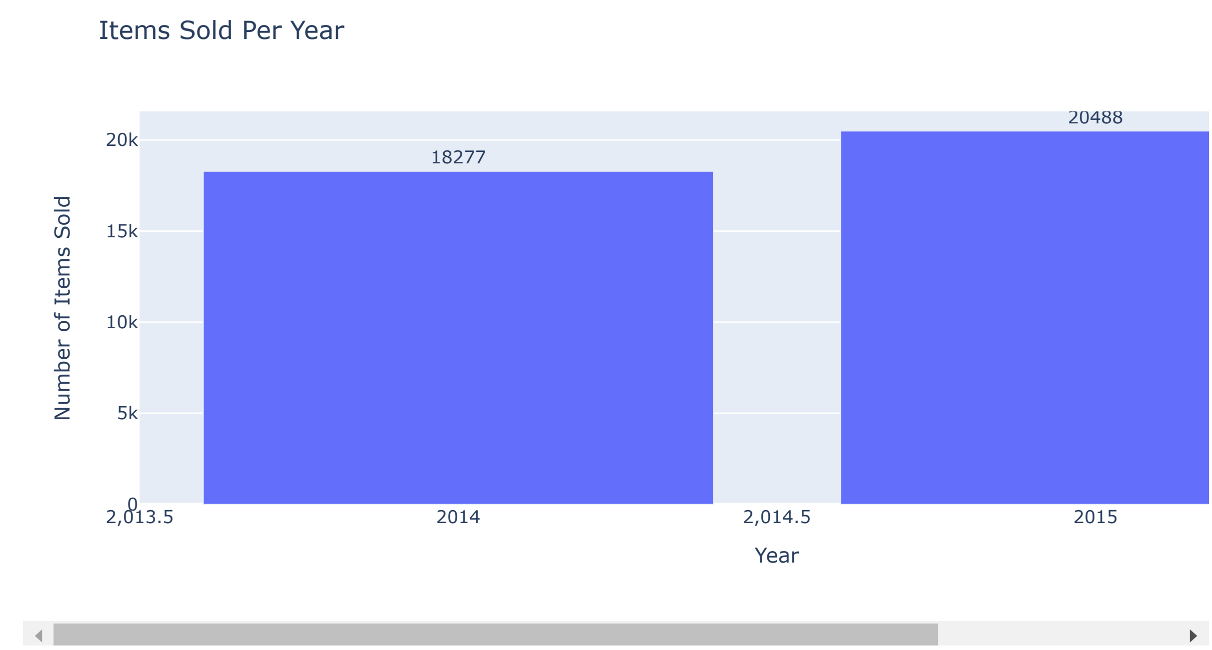
Items Sold Per Year

11

```
sns.set_style('darkgrid')
```

12

```
yearly_count = pandas_df['Year'].value_counts().reset_index()
yearly_count.columns = ['Year', 'Count']
yearly_count.sort_values(by='Year', inplace=True)

fig = px.bar(yearly_count, x='Year', y='Count',
             title='Items Sold Per Year',
             labels={'Year': 'Year', 'Count': 'Number of Items Sold'},
             text='Count')

fig.update_traces(textposition='outside')
fig.update_layout(xaxis_title='Year',
                  yaxis_title='Number of Items Sold')

fig.show()
```



13

```
pio.write_html(fig, file="Plots/Items Sold Per Year.html", auto_open=False)
```

Sales increase in 2015 as compared to 2014

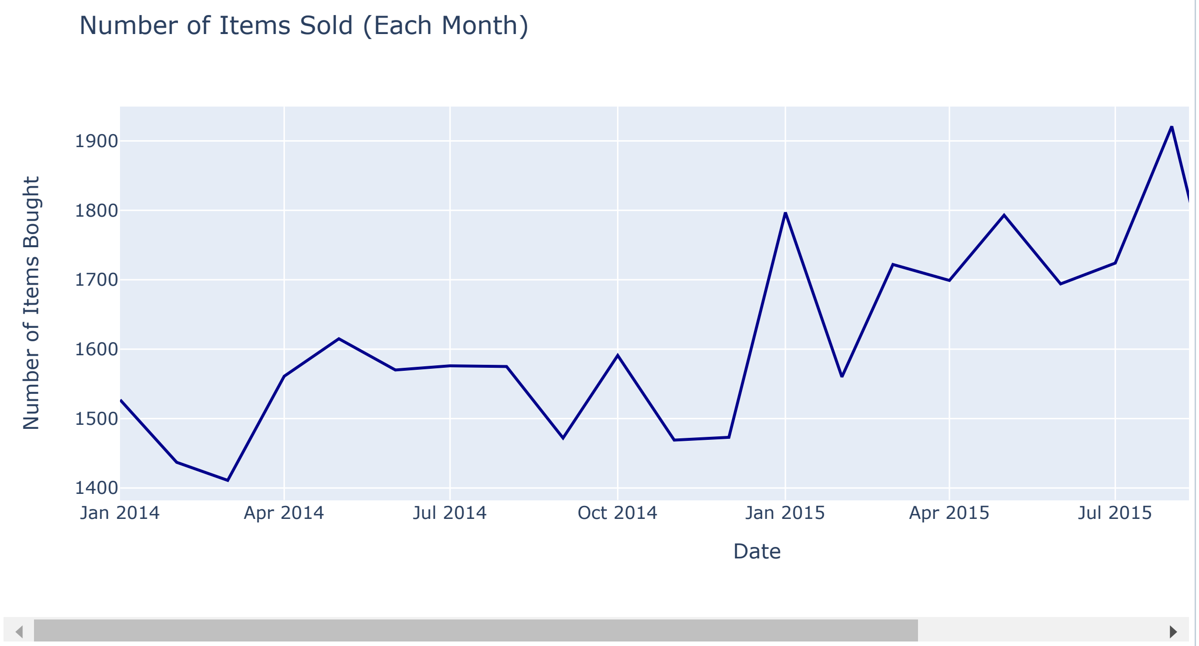
Items Sold Per Month

16

```
df_monthly = pandas_df.copy()
df_monthly['Date'] = df_monthly['Date'].apply(lambda x: pd.to_datetime(f"{x.year}/{x.month}/{1}"))
df_monthly = df_monthly.groupby('Date').count()['itemDescription'].reset_index()
```

17

```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=df_monthly['Date'],
    y=df_monthly['itemDescription'],
    mode='lines',
    line=dict(color='darkblue'),
    name='Items Sold'
))

fig.update_layout(
    title='Number of Items Sold (Each Month)',
    xaxis_title='Date',
    yaxis_title='Number of Items Bought',
    template='plotly'
)
fig.show()
```



18

```
pio.write_html(fig, file="Plots/Number of Items Sold Per Month.html", auto_open=False)
```

19

```
df_monthly = pandas_df.copy()
df_monthly.drop(['Member_number','Date','WeekOfYear'],axis=1,inplace=True)
df_monthly = df_monthly.groupby(['Year','Month']).count().reset_index()
d_2014 = df_monthly[df_monthly['Year'] == 2014]
d_2015 = df_monthly[df_monthly['Year'] == 2015]
```

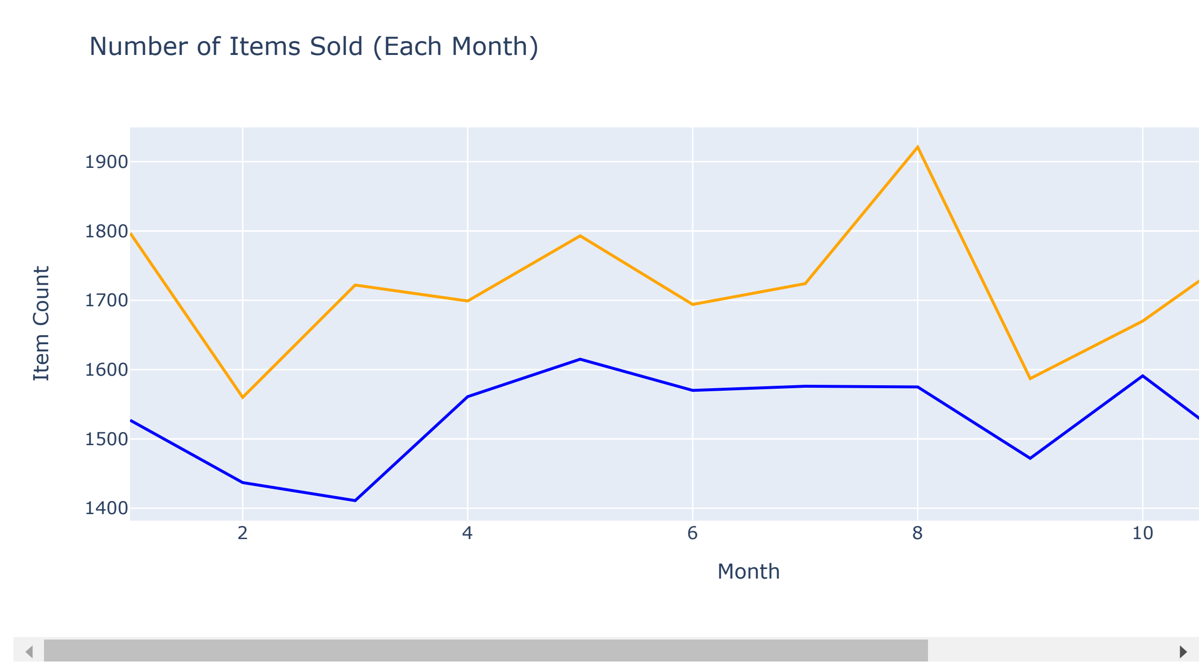
20

```
d_2014.head(12)
```

	Year	Month	itemDescription	Day
0	2014	1	1527	1527
1	2014	2	1437	1437
2	2014	3	1411	1411
3	2014	4	1561	1561
4	2014	5	1615	1615
5	2014	6	1570	1570
6	2014	7	1576	1576
7	2014	8	1575	1575
8	2014	9	1472	1472
9	2014	10	1591	1591
10	2014	11	1469	1469
11	2014	12	1473	1473

21

```
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=d_2014['Month'],
    y=d_2014['itemDescription'],
    mode='lines',
    name='2014',
    line=dict(color='blue')
))

fig.add_trace(go.Scatter(
    x=d_2015['Month'],
    y=d_2015['itemDescription'],
    mode='lines',
    name='2015',
    line=dict(color='orange')
))

fig.update_layout(
    title='Number of Items Sold (Each Month)',
    xaxis_title='Month',
    yaxis_title='Item Count',
    legend=dict(title='Year'),
    template='plotly'
)

fig.show()
```



22

```
pio.write_html(fig, file="Plots/Number of Items Sold in 2014 and 2015.html", auto_open=False)
```

23

```
corr=d_2014.merge(right=d_2015,on='Month')[['itemDescription_x','itemDescription_y']].corr().values[0][1]
print(f'Correlation between Sales in 2014 and 2015: {corr}')
```

```
Correlation between Sales in 2014 and 2015: 0.4654402963659504
```

The correlation value of 0.4654402963659504 between sales in 2014 and 2015 indicates a moderate positive correlation, it suggests that as sales in 2014 increased, sales in 2015 also tended to increase.

The sales patterns in 2014 and 2015 may be influenced by similar factors, such as market trends, customer preferences, or seasonality. However, other factors also contribute, as the correlation is not very high.

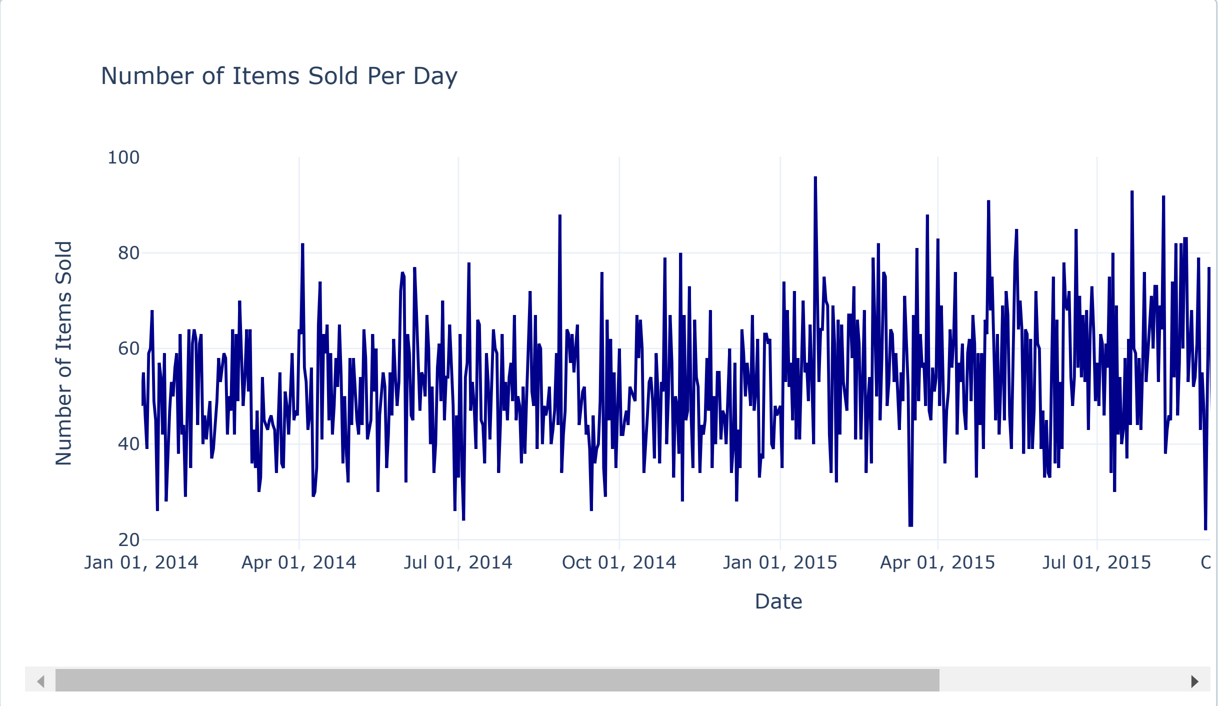
Items Sold Per Day

26

```
df_daily = pandas_df.groupby('Date').count()['itemDescription'].reset_index()
```

27

```
# Create a Plotly figure
fig = go.Figure()

# Add a trace for the daily sales
fig.add_trace(go.Scatter(
    x=df_daily['Date'],  # Date column
    y=df_daily['itemDescription'],  # Number of items sold
    mode='lines',
    line=dict(color='darkblue'),
    name='Daily Sales'
))

# Update layout for better appearance
fig.update_layout(
    title='Number of Items Sold Per Day',
    xaxis_title='Date',
    yaxis_title='Number of Items Sold',
    template='plotly_white',
    xaxis=dict(
        showgrid=True,
        tickformat='%b %d, %Y',  # Optional: Adjust date format
        title_font=dict(size=14),
    ),
    yaxis=dict(
        showgrid=True,
        title_font=dict(size=14),
    ),
    title_font=dict(size=16),
    hovermode='x'  # Hover mode aligned to x-axis
)

# Show the figure
fig.show()
```



28

```
pio.write_html(fig, file="Plots/Number of Items Sold Per Day.html", auto_open=False)
```

Observations:

1. **Fluctuating Sales**: The number of items sold daily varies significantly, ranging approximately from **20 to 100 items per day**.
2. **No Strong Trend**: There doesn't appear to be a clear upward or downward trend in the number of items sold over time. The sales seem to oscillate randomly within the range.
3. **Seasonality**: There might be some **seasonal patterns**, as occasional spikes can be observed during certain periods (e.g., beginning of 2015). However, the patterns are not immediately obvious from this graph alone.
4. **Sales Peaks**: There are noticeable spikes where sales exceed **90 items per day**. These may indicate specific events, promotions, or external factors influencing higher sales.
5. **Steady Base Level**: Despite fluctuations, the daily sales hover mostly between **40 and 70 items per day**, suggesting a relatively stable baseline demand.

Number of times each item has been sold

31

```
df_items = pandas_df.groupby('itemDescription').count().sort_values(by='Member_number',ascending=False).reset_inde
df_items.rename(columns={'itemDescription': 'Item',
                   'Member_number': 'Number of sales'},inplace=True)
df_items.drop(['Date','Year','Month','Day','WeekOfYear'],axis=1,inplace=True)
df_items.head()
```

	Item	Number of sales
0	whole milk	2502
1	other vegetables	1898
2	rolls/buns	1716
3	soda	1514
4	yogurt	1334

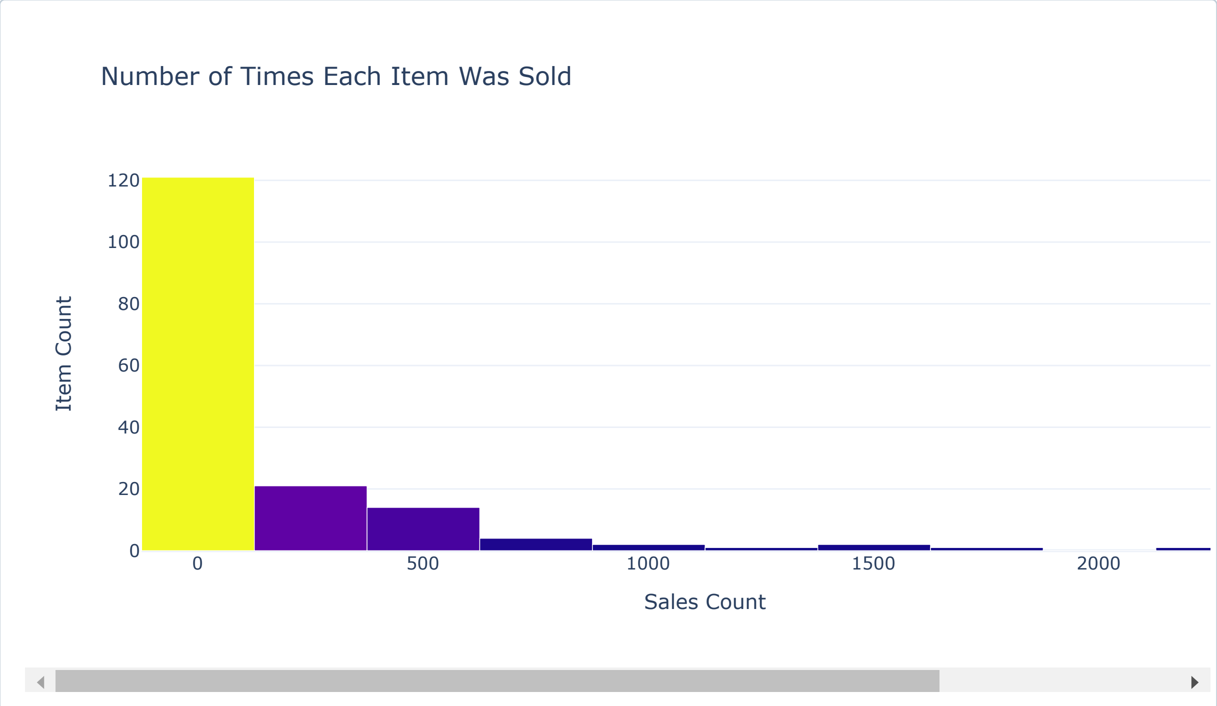
32

```
df_items['Number of sales'].describe()
```

```
count     167.000000
mean      232.125749
std       363.442098
min         1.000000
25%        30.500000
50%        85.000000
75%       264.000000
max      2502.000000
Name: Number of sales, dtype: float64
```

33

```
import numpy as np

# Calculate histogram data
counts, bins = np.histogram(df_items['Number of sales'], bins=10)

# Normalize counts for gradient coloring
norm_counts = (counts - counts.min()) / (counts.max() - counts.min())

# Create a bar chart with Plotly
fig = go.Figure()

# Add bars with gradient colors
fig.add_trace(
    go.Bar(
        x=bins[:-1],
        y=counts,
        marker=dict(
            color=norm_counts,
            colorscale='Plasma',  # Use Plasma colormap
            colorbar=dict(title="Normalized Counts")
        ),
        width=np.diff(bins),  # Set the bar width to match bin sizes
        hovertemplate='Sales Count: %{x}<br>Item Count: %{y}<extra></extra>'
    )
)

# Customize layout
fig.update_layout(
    title='Number of Times Each Item Was Sold',
    xaxis_title='Sales Count',
    yaxis_title='Item Count',
    template='plotly_white'
)

# Show the plot
fig.show()
```



34

```
pio.write_html(fig, file="Plots/Number of Times Each Item was Sold.html", auto_open=False)
```

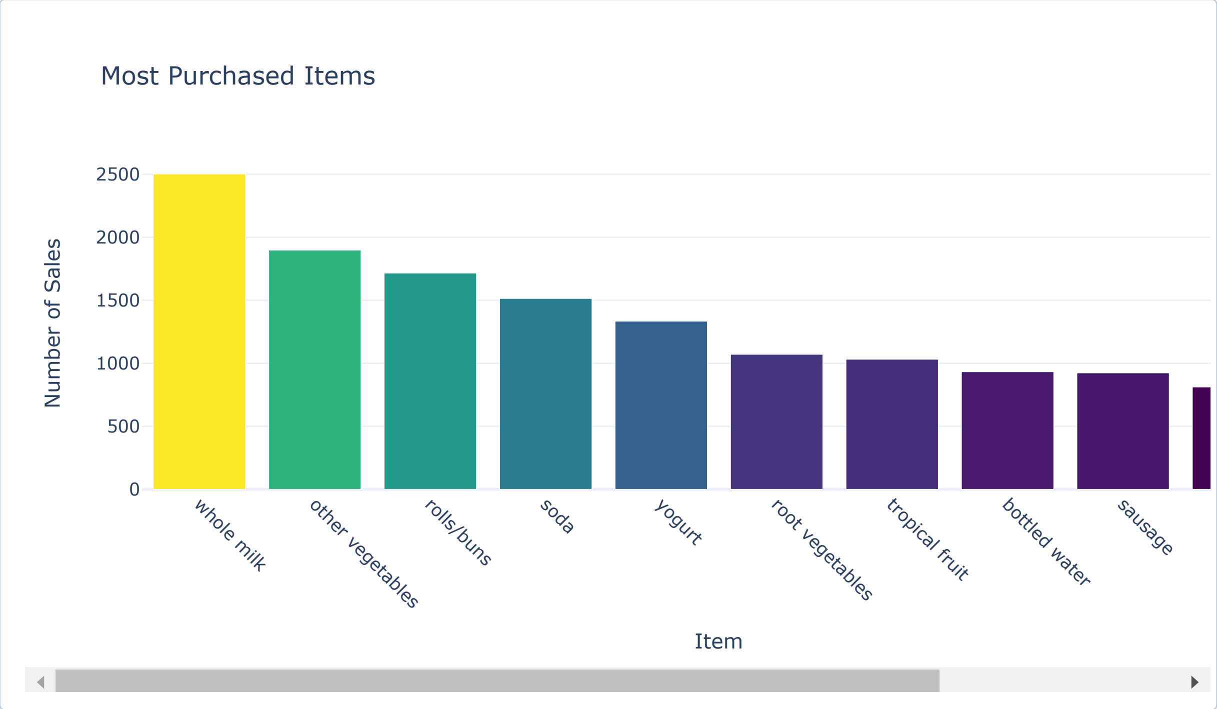
There appear to be some outliers in the data. Let's examine these outliers more closely.

36

```
df_new = pandas_df.groupby('itemDescription').count().sort_values(by='Member_number',ascending=False).head(10).res
df_new.drop(['Date','Year','Month','Day','WeekOfYear'],axis=1,inplace=True)
df_new.rename(columns={'itemDescription': 'Item',
                   'Member_number' : 'Number of sales'},inplace=True)
```

37

```
fig = px.bar(
    df_new,
    x='Item',
    y='Number of sales',
    color='Number of sales',
    color_continuous_scale='viridis',
    title='Most Purchased Items'
)

fig.update_layout(
    xaxis_title='Item',
    yaxis_title='Number of Sales',
    template='plotly_white',
    xaxis=dict(tickangle=45)
)

fig.show()
```



38

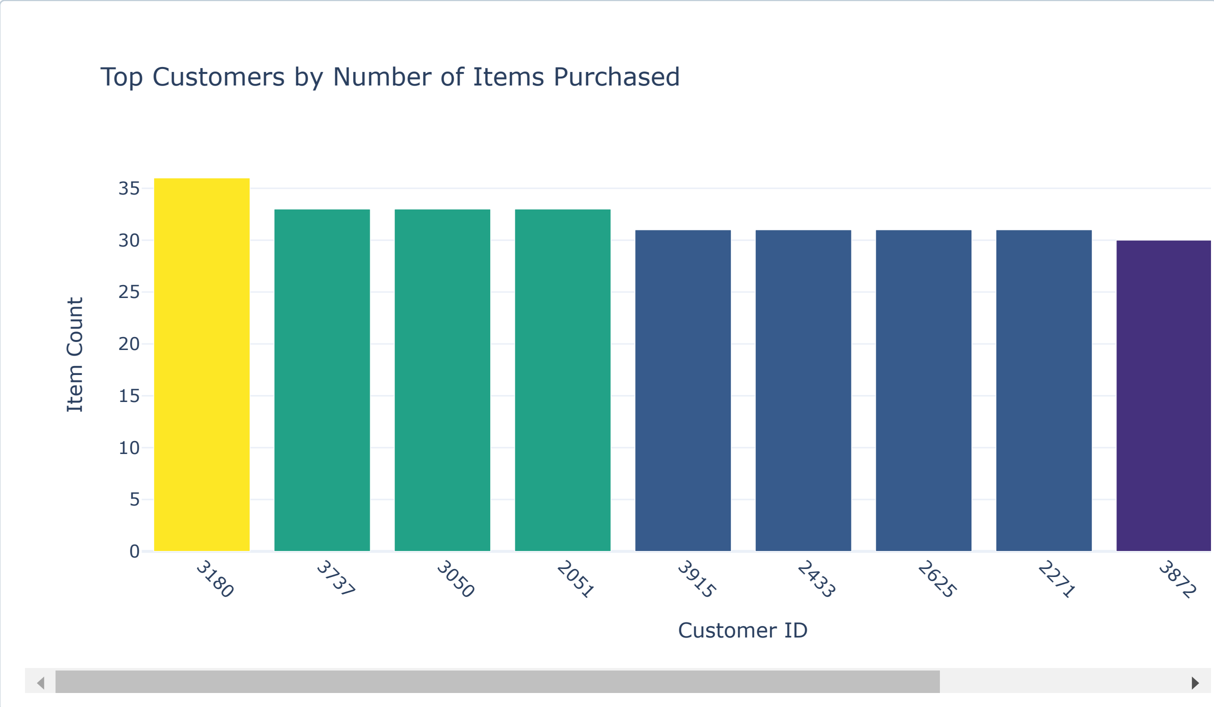
```
pio.write_html(fig, file="Plots/Most Purchased Items.html", auto_open=False)
```

39

```
df_cust = pandas_df.groupby('Member_number').count().sort_values(by='itemDescription',ascending=False).head(10).re
df_cust.drop(['Date','Year','Month','Day','WeekOfYear'],axis=1,inplace=True)
df_cust.rename(columns={'itemDescription': 'Item Count',
                   'Member_number' : 'Customer ID'},inplace=True)

df_cust['Customer ID'] = df_cust['Customer ID'].astype(str)
```

40

```
fig = px.bar(
    df_cust,
    x='Customer ID',
    y='Item Count',
    color='Item Count',
    color_continuous_scale='viridis',
    title='Top Customers by Number of Items Purchased'
)

fig.update_layout(
    xaxis_title='Customer ID',
    yaxis_title='Item Count',
    template='plotly_white',
    xaxis=dict(tickangle=45)
)

fig.show()
```



41

```
pio.write_html(fig, file="Plots/Top Customers by Number of Items Purchased.html", auto_open=False)
```

Currently, our data is structured in a way that prevents us from determining how many items (and which specific items) each customer purchased during each visit to the store. For instance, consider the following table

43

```
pandas_df[pandas_df['Member_number'] == 4875].sort_values(by='Date').head()
```

	Member_number	Date	itemDescription	Year	Month	Day	WeekOfYear
37401	4875	2014-04-15	salt	2014	4	15	16
34084	4875	2014-04-15	misc. beverages	2014	4	15	16
30103	4875	2014-04-15	bottled beer	2014	4	15	16
13954	4875	2014-04-15	rolls/buns	2014	4	15	16
29681	4875	2014-06-05	chocolate	2014	6	5	23

We can see that the customer with ID 4875 purchased 4 items on April 04, 2014. However, the issue is that we don't know how many times this customer visited the store on that day, nor what items they bought during each visit. It's possible that they visited the store 4 times, purchasing one item each time. Alternatively, they might have visited twice. Or perhaps they visited just once and bought all 4 items together. Without this information, we cannot determine the exact scenario. For meaningful association analysis, though, we need this level of detail. Since the available data does not provide this information, we will make the following assumption:

Assumption: Each customer visited the store **only once per day**."

Creating Transactions Document

46

```
pandas_df.head()
```

	Member_number	Date	itemDescription	Year	Month	Day	WeekOfYear
0	1808	2015-07-21	tropical fruit	2015	7	21	30
1	2552	2015-01-05	whole milk	2015	1	5	2
2	2300	2015-09-19	pip fruit	2015	9	19	38
3	1187	2015-12-12	other vegetables	2015	12	12	50
4	3037	2015-02-01	whole milk	2015	2	1	5

47

```
pandas_df.iloc[:,0:3]
```

	Member_number	Date	itemDescription
0	1808	2015-07-21	tropical fruit
1	2552	2015-01-05	whole milk
2	2300	2015-09-19	pip fruit
3	1187	2015-12-12	other vegetables
4	3037	2015-02-01	whole milk
...	...	...	...
38760	4471	2014-10-08	sliced cheese
38761	2022	2014-02-23	candy
38762	1097	2014-04-16	cake bar
38763	1510	2014-12-03	fruit/vegetable juice
38764	1521	2014-12-26	cat food

38765 rows × 3 columns

48

```
df1 = pandas_df.iloc[:,0:3].copy()

df1['itemDescription'] = df1['itemDescription'].apply(lambda x: [x,]).copy()
df1 = df1.groupby(['Member_number','Date']).agg(sum).reset_index()
df1.rename(columns={'itemDescription': 'Items_Bought'},inplace=True)
df1.head()
```

	Member_number	Date	Items_Bought
0	1000	2014-06-24	[whole milk, pastry, salty snack]
1	1000	2015-03-15	[sausage, whole milk, semi-finished bread, yog...
2	1000	2015-05-27	[soda, pickled vegetables]
3	1000	2015-07-24	[canned beer, misc. beverages]
4	1000	2015-11-25	[sausage, hygiene articles]

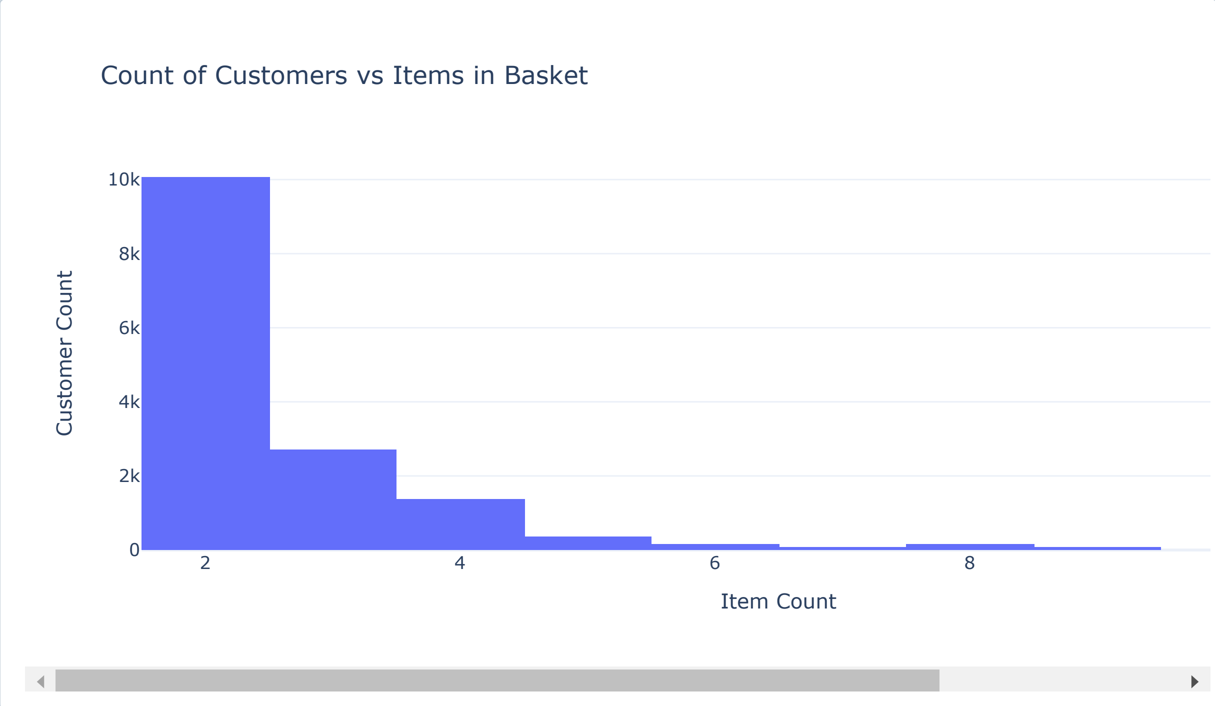
Items_Bought represents the set of all items purchased during a visit by a customer

50

```
df1['Basket size'] = df1['Items_Bought'].apply(lambda x: len(x))
```

51

```
fig = px.histogram(
    df1,
    x='Basket size',
    title="Count of Customers vs Items in Basket"
)

fig.update_layout(
    xaxis_title='Item Count',
    yaxis_title='Customer Count',
    template='plotly_white'
)

fig.show()
```



52

```
pio.write_html(fig, file="Plots/Count of Customers vs Items in Basket.html", auto_open=False)
```

Next, we will apply Association Rule Learning to explore potential patterns in customers' purchasing behavior. To generate a set of meaningful rules, we will use two popular algorithms: the Apriori algorithm and FP-growth.

54

```
from mlxtend.frequent_patterns import apriori, fpgrowth
from mlxtend.frequent_patterns import association_rules
import mlxtend as ml
```

55

```
df1 = pandas_df.iloc[:,0:3].copy()
df1['itemDescription'] = df1['itemDescription'].apply(lambda x: [x,]).copy()
df1 = df1.groupby(['Member_number','Date']).agg(sum).reset_index()
df1.rename(columns={'itemDescription': 'Items_Bought'},inplace=True)
df1.head()
```

	Member_number	Date	Items_Bought
0	1000	2014-06-24	[whole milk, pastry, salty snack]
1	1000	2015-03-15	[sausage, whole milk, semi-finished bread, yog...
2	1000	2015-05-27	[soda, pickled vegetables]
3	1000	2015-07-24	[canned beer, misc. beverages]
4	1000	2015-11-25	[sausage, hygiene articles]

56

```
all_items = pandas_df['itemDescription'].unique()
data = []


for transaction in df1['Items_Bought']:
    row = []
    for item in all_items:
        if item in transaction:
            row.append(1)
        else:
            row.append(0)
    data.append(row)

df2 = pd.DataFrame(data,columns=all_items)
df2 = df2.rename_axis('Transcation ID')
```

57

```
df2.head()
```

Transcation ID	tropical fruit	whole milk	pip fruit	other vegetables	rolls/buns	pot plants	citrus fruit	beef	frankfurter	chicken	butter	fruit/vegetable juice	pac fruit/vege
0	0	1	0	0	0	0	0	0	0	0	0	0	
1	0	1	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 167 columns

Applying Apriori Algorithm

59

```
frequent_itemsets = apriori(df2, min_support=0.001, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift", num_itemsets=2)
rules.sort_values('confidence', ascending = False, inplace = True)
```

```
/local_disk0/.ephemeral_nfs/cluster_libraries/python/lib/python3.11/site-packages/mlxtend/frequent_patterns/fpcommo
n.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their supp
ort might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

60

```
rules = rules[rules['confidence'] > 0.1].copy()
rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zha
718	(yogurt, sausage)	(whole milk)	0.005748	0.157923	0.001470	0.255814	1.619866	1.0	0.000563	1.131541	
712	(sausage, rolls/buns)	(whole milk)	0.005347	0.157923	0.001136	0.212500	1.345594	1.0	0.000292	1.069304	
724	(sausage, soda)	(whole milk)	0.005948	0.157923	0.001069	0.179775	1.138374	1.0	0.000130	1.026642	
125	(semi-finished bread)	(whole milk)	0.009490	0.157923	0.001671	0.176056	1.114825	1.0	0.000172	1.022008	
706	(yogurt, rolls/buns)	(whole milk)	0.007819	0.157923	0.001337	0.170940	1.082428	1.0	0.000102	1.015701	

61

```
rows = rules.shape[0]
print(f'Number of rules: {rows}')
```

```
Number of rules: 99
```

We observe that the support for all the rules in our dataset is quite low, meaning the proportion of transactions that include items from both baskets is minimal. This could pose a challenge, as any results derived from this analysis may not be statistically significant.

Rules with the highest lift

64

```
rules.sort_values(by='lift',ascending=False).head(10).iloc[:,:-2][['antecedents',
                                                                  'consequents',
                                                                   'consequent support',
                                                                  'lift']]
```

	antecedents	consequents	consequent support	lift
716	(whole milk, yogurt)	(sausage)	0.060349	2.182917
717	(whole milk, sausage)	(yogurt)	0.085879	1.911760
718	(yogurt, sausage)	(whole milk)	0.157923	1.619866
20	(flour)	(tropical fruit)	0.067767	1.617141
561	(processed cheese)	(root vegetables)	0.069572	1.513019
493	(soft cheese)	(yogurt)	0.085879	1.474952
471	(detergent)	(yogurt)	0.085879	1.444261
488	(chewing gum)	(yogurt)	0.085879	1.358508
712	(sausage, rolls/buns)	(whole milk)	0.157923	1.345594
248	(processed cheese)	(rolls/buns)	0.110005	1.315734

We observe that the itemsets (yogurt, whole milk) and (sausage) have the highest lift, which means that once a customer buys yogurt and whole milk, it becomes 2.2 times more likely that they will also purchase sausage. However, as we've noted, due to the low support, it's difficult to determine whether this is a genuine association or just a random occurrence.

Similarly, we will examine the rules with the lowest lift, where the items in the antecedent and consequent are less likely to be bought together.

66

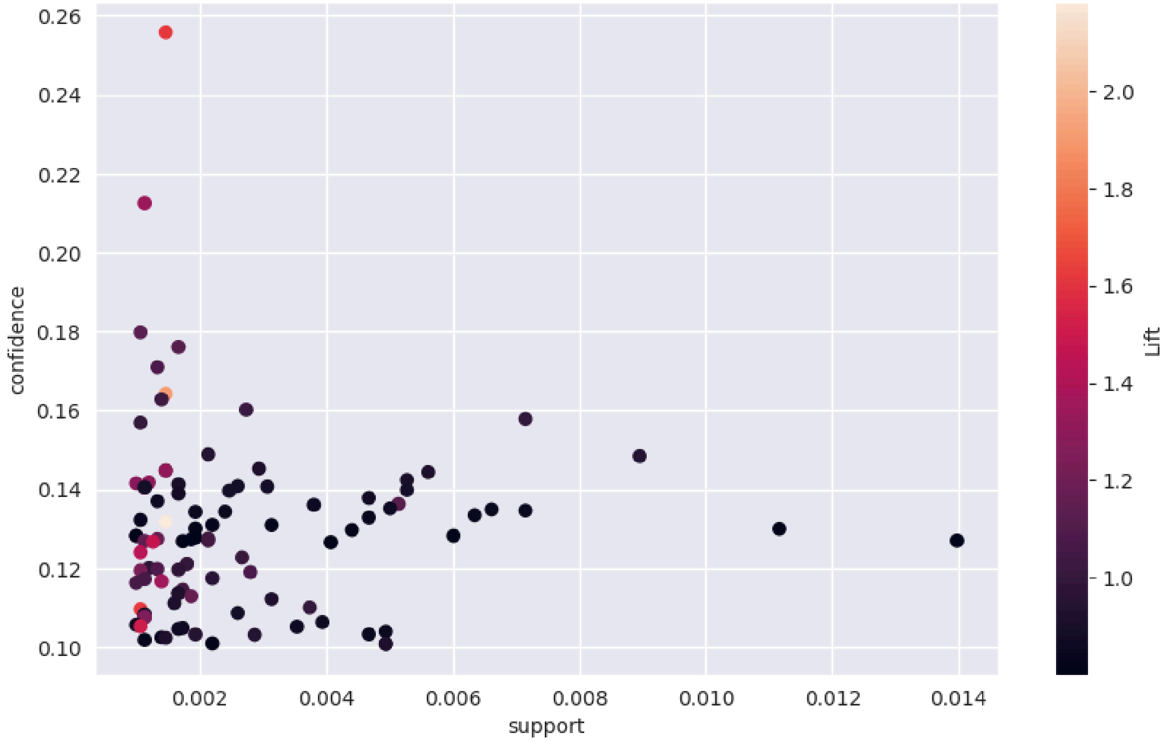
```
rules.sort_values(by='lift',ascending=True).head(10).iloc[:,:-2][['antecedents',
                                                                 'consequents',
                                                                  'consequent support',
                                                                 'lift']]
```

	antecedents	consequents	consequent support	lift
109	(margarine)	(whole milk)	0.157923	0.801379
121	(hygiene articles)	(whole milk)	0.157923	0.803109
55	(rolls/buns)	(whole milk)	0.157923	0.804028
101	(hard cheese)	(whole milk)	0.157923	0.805917
99	(ice cream)	(whole milk)	0.157923	0.808960
77	(canned beer)	(whole milk)	0.157923	0.811821
57	(pot plants)	(whole milk)	0.157923	0.811821
67	(fruit/vegetable juice)	(whole milk)	0.157923	0.821072
69	(yogurt)	(whole milk)	0.157923	0.822940
113	(oil)	(whole milk)	0.157923	0.823471

Visualizing the relation between support, confidence and lift

68

```
sup = rules['support'].values
conf = rules['confidence'].values
lift = rules['lift'].values

plt.figure(figsize=(10,6))
sc = plt.scatter(sup,conf,c=lift)
plt.colorbar(sc,label='Lift')
plt.xlabel('support')
plt.ylabel('confidence')
plt.show()
```



We can make the following observations:

1. **Low Support with Higher Confidence:** Most of the rules are clustered in the lower-left section of the graph, indicating low support values. However, their confidence tends to be slightly higher, with most values lying between 0.10 and 0.14. This suggests that while the rules are not frequent (low support), when they do occur, they have a reasonable chance of being accurate (moderate confidence).

2. **Outliers with High Confidence:** There is one outlier in the top left corner with high confidence (~0.26), which indicates that this rule is highly confident but still has low support. This could point to a potentially interesting association, but its low support makes it less reliable.

3. **Lift:** The colors in the graph represent lift, with darker colors corresponding to lower lift values (around 1). Lift values for most of the points are below 1.5, indicating that the associations are not strongly significant. Only a few points have a higher lift (up to 2), meaning that for these specific rules, the items appear more frequently together than would be expected by chance.

Conclusion:

- Most associations identified are weak, with low support and only moderate confidence.
- A few potential strong rules (with higher confidence and lift) stand out, but these need to be treated cautiously due to the low support.
- The overall data suggests that while there might be associations, they are not very robust and may not be

Applying FP Growth Algorithm

71

```
frequent_itemsets = fpgrowth(df2, min_support=0.001, use_colnames=True)
```

```
/local_disk0/.ephemeral_nfs/cluster_libraries/python/lib/python3.11/site-packages/mlxtend/frequent_patterns/fpcommo
n.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their supp
ort might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

72

```
fp_rules = association_rules(frequent_itemsets, num_itemsets=10, metric="confidence", min_threshold=0.1)
```

73

```
fp_rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	representativity	leverage	conviction	zhan
0	(pastry)	(whole milk)	0.051728	0.157923	0.006483	0.125323	0.793571	1.0	-0.001686	0.962729	
1	(salty snack)	(whole milk)	0.018780	0.157923	0.001938	0.103203	0.653502	1.0	-0.001028	0.938983	
2	(salty snack)	(rolls/buns)	0.018780	0.110005	0.001938	0.103203	0.938168	1.0	-0.000128	0.992415	
3	(salty snack)	(other vegetables)	0.018780	0.122101	0.002205	0.117438	0.961807	1.0	-0.000088	0.994716	
4	(yogurt)	(whole milk)	0.085879	0.157923	0.011161	0.129961	0.822940	1.0	-0.002401	0.967861	

74

```
len(fp_rules)
```

```
130
```

Again the support is very low.

76

```
fp_rules.sort_values(by='lift',ascending=False).head(10).iloc[:,:-2][['antecedents',
                                                                   'consequents',
                                                                    'consequent support',
                                                                   'lift']]
```

	antecedents	consequents	consequent support	lift
10	(whole milk, yogurt)	(sausage)	0.060349	2.182917
11	(whole milk, sausage)	(yogurt)	0.085879	1.911760
12	(yogurt, sausage)	(whole milk)	0.157923	1.619866
70	(flour)	(tropical fruit)	0.067767	1.617141
93	(processed cheese)	(root vegetables)	0.069572	1.513019
103	(soft cheese)	(yogurt)	0.085879	1.474952
49	(detergent)	(yogurt)	0.085879	1.444261
127	(chewing gum)	(yogurt)	0.085879	1.358508
14	(sausage, rolls/buns)	(whole milk)	0.157923	1.345594
92	(processed cheese)	(rolls/buns)	0.110005	1.315734

We observed similar results in apriori

Conclusion:

Both the Apriori and FP-Growth algorithms yield similar results, but the Apriori algorithm generates less association rules compared to FP-Growth. However, the robustness of these rules is limited due to the low support values in the dataset, indicating that the associations identified may not be statistically significant. Consequently, while the algorithms provide insights into potential associations, the results should be interpreted with caution.

79

```
rules.to_csv('Data/Apriori_rules.csv', index=False)
```

80