In [1]:
```
# Importing necessary libraries

import pandas as pd
import numpy as np
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
from mlxtend.frequent_patterns import apriori, association_rules
```

In [2]:
```
df = pd.read_csv(".//Dataset//market_basket_dataset.csv")
df
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[2]:

	BillNo	Itemname	Quantity	Price	CustomerID
0	1000	Apples	5	8.30	52299
1	1000	Butter	4	6.06	11752
2	1000	Eggs	4	2.66	16415
3	1000	Potatoes	4	8.10	22889
4	1004	Oranges	2	7.26	52255
...	...	...	...	...	...
495	1493	Juice	2	4.24	55321
496	1493	Bread	5	7.05	14479
497	1497	Coffee	3	2.01	25378
498	1497	Pasta	3	2.64	53334
499	1497	Eggs	4	7.37	34687

500 rows × 5 columns

In [3]:
```
# Shiftng  column 'CustomerID' to first position
first_column = df.pop('CustomerID')

df.insert(0, 'CustomerID', first_column)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [4]:
```
# Check for null values

print(df.isnull().sum())
```

```
CustomerID    0
BillNo        0
Itemname      0
Quantity      0
Price         0
dtype: int64

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

In [5]:
```
# Statistics of dataset

df.describe()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[5]:

	CustomerID	BillNo	Quantity	Price
count	500.000000	500.000000	500.000000	500.000000
mean	54229.800000	1247.442000	2.978000	5.617660
std	25672.122585	144.483097	1.426038	2.572919
min	10504.000000	1000.000000	1.000000	1.040000
25%	32823.500000	1120.000000	2.000000	3.570000
50%	53506.500000	1246.500000	3.000000	5.430000
75%	76644.250000	1370.000000	4.000000	7.920000
max	99162.000000	1497.000000	5.000000	9.940000

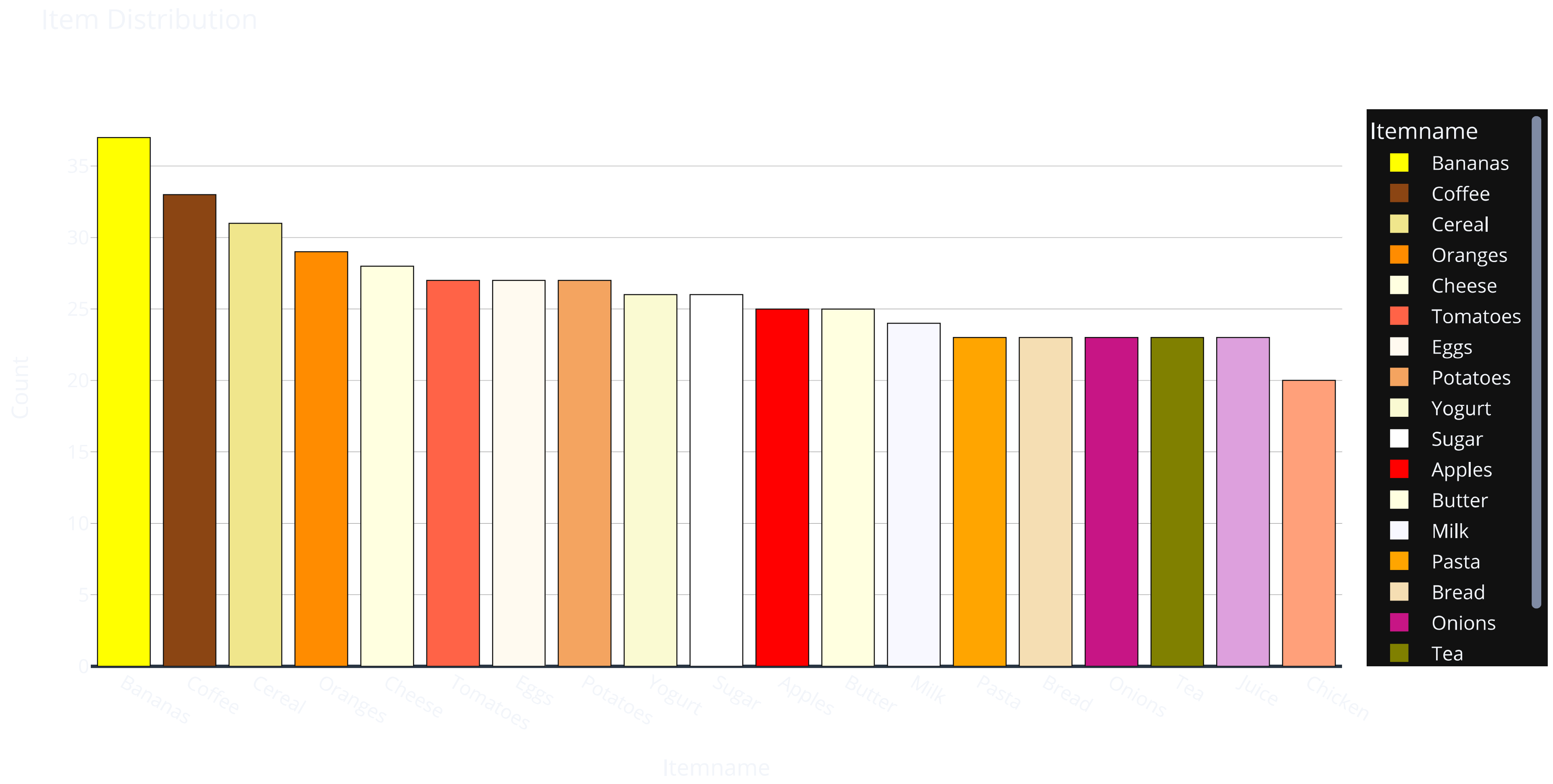
In [6]:
```
df["Itemname"].unique()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

Out[6]:
```
array(['Apples', 'Butter', 'Eggs', 'Potatoes', 'Oranges', 'Milk',
       'Onions', 'Cereal', 'Tomatoes', 'Bananas', 'Pasta', 'Bread',
       'Coffee', 'Sugar', 'Chicken', 'Cheese', 'Tea', 'Yogurt', 'Juice'],
      dtype=object)
```

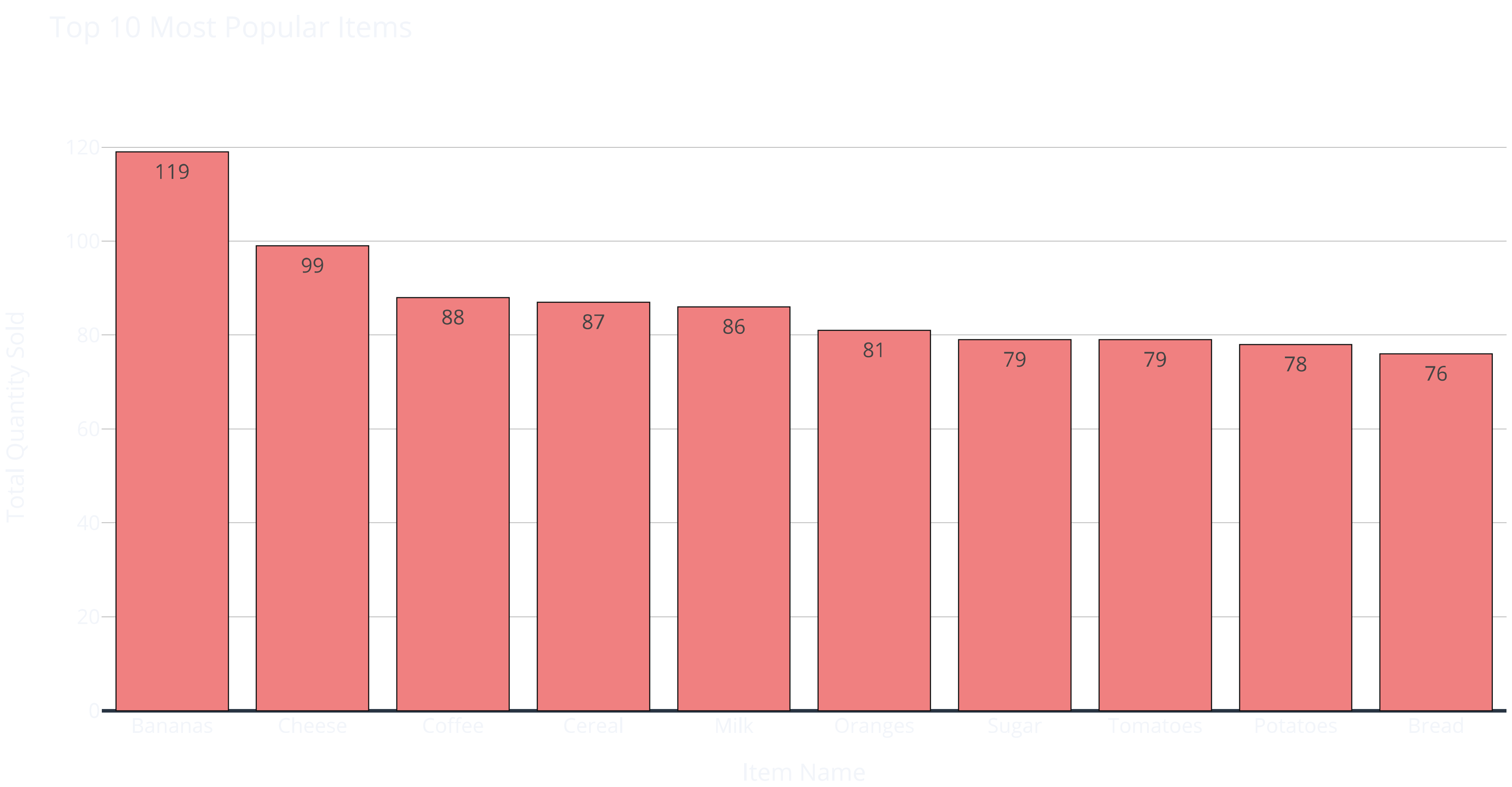
In [7]:
```python
# Items count distribution

pio.templates.default = "plotly_dark"
custom_color = {
    'Bananas':'yellow', 'Coffee':'saddlebrown','Cereal': 'khaki', 'Oranges': 'darkorange', 'Cheese':'lightyellow',
    'Tomatoes':'tomato','Eggs': 'floralwhite','Potatoes':'sandybrown','Sugar':'white','Yogurt':'lightgoldenrodyellow',
    'Apples':'Red','Butter':"lightyellow", 'Milk':'ghostwhite','Tea':'olive', 'Juice':'plum',
    'Onions':'mediumvioletred','Pasta':'orange', 'Bread':'wheat','Chicken':'lightsalmon'
}

item_count= df["Itemname"].value_counts().reset_index()
item_count.columns = ['Itemname', 'Count']
fig = px.bar( item_count, x = "Itemname" , y = "Count", color= "Itemname", color_discrete_map=custom_color, title="Item Distribution")



fig.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipykernel.py:287: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please p
ass the result to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



In [8]:
```python
# Sales of top 10 Items

item_sales= df.groupby("Itemname")["Quantity"].sum().sort_values(ascending=False)

top_n = 10
fig = go.Figure()
fig.add_trace(go.Bar(x=item_sales.index[:top_n], y=item_sales.values[:top_n],
                     text=item_sales.values[:top_n], textposition='auto',
                     marker=dict(color='lightcoral')))
fig.update_layout(title=f'Top {top_n} Most Popular Items',
                  xaxis_title='Item Name', yaxis_title='Total Quantity Sold')
fig.show()
```

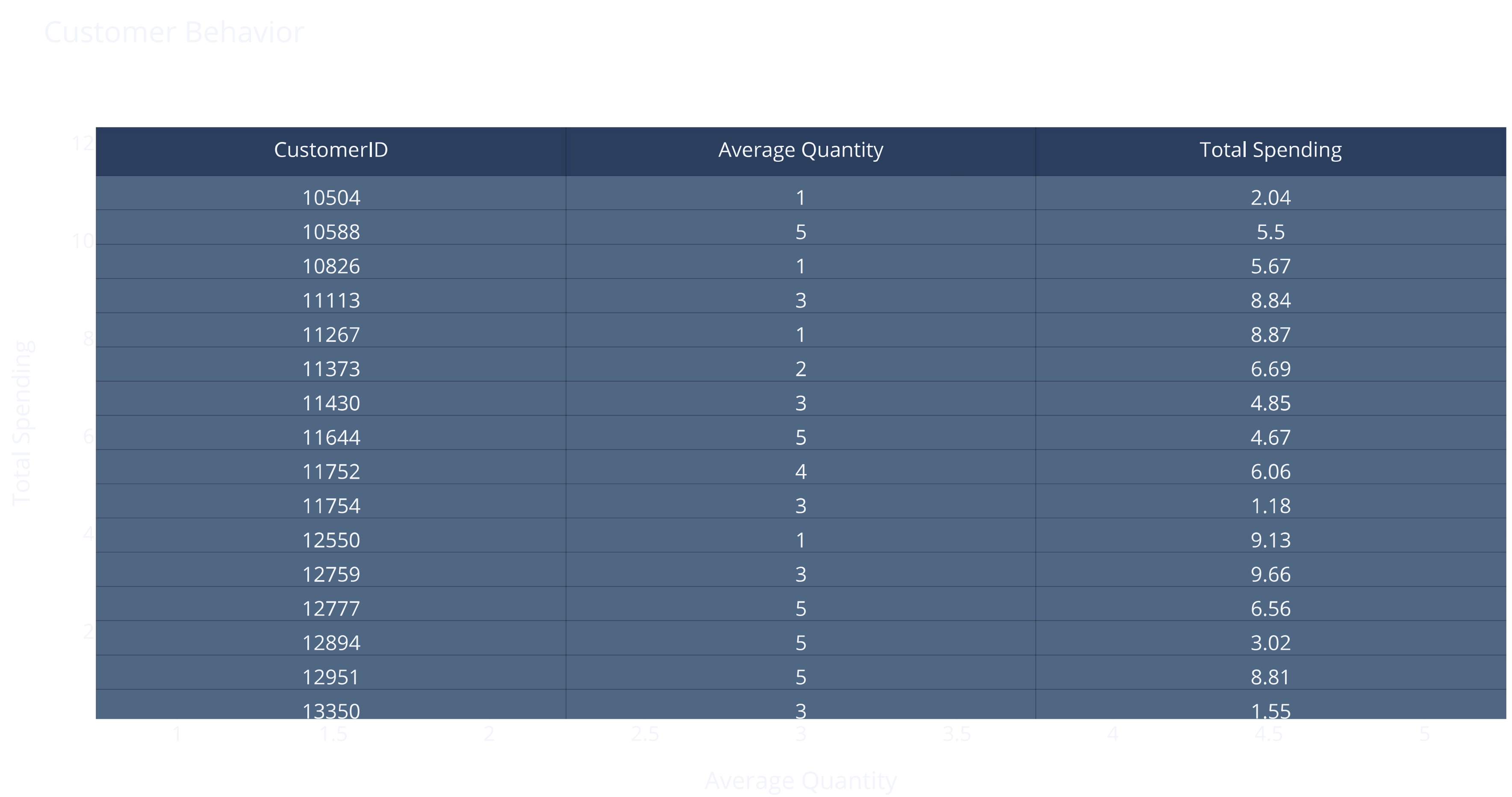
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetran
sform in `preprocessing_exc_tuple` in IPython 7.17 and above.
```



From above graph, it is evident bananas are the most popular items sold.

In [9]:
```
# Calculate average quantity and spending per customer
customer_behavior = df.groupby('CustomerID').agg({'Quantity': 'mean', 'Price': 'sum'}).reset_index()

# Create a DataFrame to display the values
df_new = pd.DataFrame({
    'CustomerID': customer_behavior['CustomerID'],
    'Average Quantity': customer_behavior['Quantity'],
    'Total Spending': customer_behavior['Price']
})

# Create a subplot with a scatter plot and a table
fig = go.Figure()

# Add a scatter plot
fig.add_trace(go.Scatter(x=customer_behavior['Quantity'], y=customer_behavior['Price'],
                         mode='markers', text=customer_behavior['CustomerID'],
                         marker=dict(size=10, color='coral')))

# Add a table
fig.add_trace(go.Table(
    header=dict(values=['CustomerID', 'Average Quantity', 'Total Spending']),
    cells=dict(values=[df_new['CustomerID'], df_new['Average Quantity'], df_new['Total Spending']]),
))

# Update layout
fig.update_layout(title='Customer Behavior',
                  xaxis_title='Average Quantity', yaxis_title='Total Spending')

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

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetran
sform in `preprocessing_exc_tuple` in IPython 7.17 and above.
```



CustomerID	Average Quantity	Total Spending
10504	1	2.04
10588	5	5.5
10826	1	5.67
11113	3	8.84
11267	1	8.87
11373	2	6.69
11430	3	4.85
11644	5	4.67
11752	4	6.06
11754	3	1.18
12550	1	9.13
12759	3	9.66
12777	5	6.56
12894	5	3.02
12951	5	8.81
13350	3	1.55

Utilize the Apriori algorithm for generating association rules. This algorithm is employed to detect frequent item sets within extensive transactional datasets, aiming to pinpoint items frequently bought together. By unveiling patterns in customer behavior, it enables businesses to make well-informed decisions regarding product placement, promotional activities, and marketing strategies

In [10]:
```
# Group items by BillNo and create a list of items for each bill
basket = df.groupby('BillNo')['Itemname'].apply(list).reset_index()

# Encode items as binary variables using one-hot encoding
basket_encoded = basket['Itemname'].str.join('|').str.get_dummies('|')

# Find frequent itemsets using Apriori algorithm with lower support
frequent_itemsets = apriori(basket_encoded, min_support=0.01, use_colnames=True)

# Generate association rules with lower lift threshold
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=0.5)

# Display association rules
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(10))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning:

`should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any exception that happen during thetran
sform in `preprocessing_exc_tuple` in IPython 7.17 and above.

C:\Users\komal\AppData\Roaming\Python\Python38\site-packages\mlxtend\frequent_patterns\fpcommon.py:109: DeprecationWarning:

DataFrames with non-bool types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type


  antecedents consequents   support  confidence      lift
0     (Bread)    (Apples)  0.045752    0.304348  1.862609
1    (Apples)     (Bread)  0.045752    0.280000  1.862609
2    (Apples)    (Butter)  0.026144    0.160000  0.979200
3    (Butter)    (Apples)  0.026144    0.160000  0.979200
4    (Apples)    (Cereal)  0.019608    0.120000  0.592258
5    (Cereal)    (Apples)  0.019608    0.096774  0.592258
6    (Apples)    (Cheese)  0.039216    0.240000  1.311429
7    (Cheese)    (Apples)  0.039216    0.214286  1.311429
8    (Apples)   (Chicken)  0.032680    0.200000  1.530000
9   (Chicken)    (Apples)  0.032680    0.250000  1.530000
```

The provided output displays association rules depicting relationships between various items (antecedents) and those commonly purchased alongside them (consequents).

Antecedents: These represent the initial items or the "if" part of the association rule. For instance, in this analysis, Bread, Butter, Cereal, Cheese, and Chicken are identified as antecedents.

Consequents: These items are frequently bought together with the antecedents, constituting the "then" part of the association rule.

Support: This metric gauges how often a specific combination of items (both antecedents and consequents) appears in the dataset. It signifies the proportion of transactions where the items are purchased together. For example, the first rule suggests that Bread and Apples are bought together in roughly 4.58% of transactions.

Confidence: Confidence measures the likelihood of purchasing the consequent item when the antecedent item is already present in the basket. It indicates the probability of buying the consequent item given the presence of the antecedent item. For instance, the first rule indicates a 30.43% chance of purchasing Apples when Bread is already in the basket.

Lift: Lift assesses the strength of association between the antecedent and consequent items, relative to the baseline probability of purchasing the consequent item independently. A lift value exceeding 1 signifies a positive association, indicating that the items are more likely to be purchased together than separately. Conversely, a value below 1 suggests a negative association. For instance, the first rule exhibits a lift of approximately 1.86, indicating a positive association between Bread and Apples.

In [ ]: