## ASSOCIATION RULE MINING

---

### Exploratory Data Analysis (EDA)

- Import necessary libraries and suppress warnings

This assignment utilises Jupyter Notebook Python 3 to implement the solution. Firstly, the necessary libraries are being set up for data analysis, preprocessing and visualisation. The filterwarnings function is also being import to suppress warnings from being displayed in the output.

```
# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.style
import matplotlib.pyplot as plt

# Suppress warnings
from warnings import filterwarnings
filterwarnings("ignore")
```

- Read data from CSV file

Read data from a CSV file named “Dataset_A.csv” and load it into a Pandas dataframe called ‘df’ for further analysis and manipulation. The size of the dataset is 19351 rows by 3 columns.

```
# Read data from a CSV file
df = pd.read_csv("Dataset_A.csv")
```

```
df
```

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

19351 rows × 3 columns

- Rename columns

Rename the columns of ‘df’ to ‘Member_id’, ‘Date’, and ‘Item’ clarity and consistency.

```
# Rename the columns of DataFrame
df.columns = ['Member_id', 'Date', 'Item']
```

- Check on missing values

The dataset is clear from any missing values.

```
# Check on missing values
df.isna().sum()
```

```
Member_id    0
Date         0
Item         0
dtype: int64
```

- Summary of dataset

This code provides a concise summary of the data frame which provides information about the data frame structure, including the number of rows and columns, column names, data types of columns and the memory usage.

```
# Display the summary of dataset
df.info()
```

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

- Convert datatype of ‘Date’ column

Converts the ‘Date’ column from object data type to a datetime format, allowing for date-based operations and analysis.

```
# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])
```

- Extract time-related attributes

Add new columns containing the year, month, and day information, which can be useful for further analysis and grouping the data based on these time-related attributes.

```
# Extract year, month and day and create new columns
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
```

```
df
```

| | Member_id | Date | Item | Year | Month | Day |
|---|---|---|---|---|---|---|
| **0** | 1808 | 2015-07-21 | tropical fruit | 2015 | 7 | 21 |
| **1** | 2552 | 2015-05-01 | whole milk | 2015 | 5 | 1 |
| **2** | 2300 | 2015-09-19 | pip fruit | 2015 | 9 | 19 |
| **3** | 1187 | 2015-12-12 | other vegetables | 2015 | 12 | 12 |
| **4** | 3037 | 2015-01-02 | whole milk | 2015 | 1 | 2 |
| **...** | ... | ... | ... | ... | ... | ... |
| **19346** | 3478 | 2015-02-04 | rolls/buns | 2015 | 2 | 4 |
| **19347** | 2295 | 2015-12-28 | pip fruit | 2015 | 12 | 28 |
| **19348** | 1833 | 2015-07-24 | dessert | 2015 | 7 | 24 |
| **19349** | 3307 | 2015-03-03 | other vegetables | 2015 | 3 | 3 |
| **19350** | 3562 | 2015-03-18 | salty snack | 2015 | 3 | 18 |

19351 rows × 6 columns

- Unique values of ‘Item’

Display the number of occurrences of each unique value in the 'Item' column which provides a frequency distribution of items in the dataset. Whole milk is the most bought item from the store, while rubbing alcohol, kitchen utensil and specialty vegetables are the least bought items with only one occurrence each.

```
# Unique values in 'Item' column
df['Item'].value_counts()
```

```
whole milk              1545
other vegetables        1146
sausage                  924
tropical fruit           879
rolls/buns               815
                        ...
salad dressing             2
pudding powder             2
rubbing alcohol            1
kitchen utensil            1
specialty vegetables       1
Name: Item, Length: 163, dtype: int64
```

- Summary statistic

Describe the summary statistics for the frequency of unique items products in the dataset. The statistics can be summarise as follows:

- There are a total of 163 unique items in the store.
- The minimum frequency among the unique items is 1. This is the least frequently purchased item in the dataset.
- The average frequency of the unique items is approximately 118.72 indicates that on average, each item is purchased around 119 times.

- The standard deviation of the item frequencies is approximately 226.13. This measure of variability indicates the extent to which the item frequencies deviate from the mean.
- The 25th percentile of item frequencies is 8. 25% of the unique items have a frequency of 8 or less.
- The median or 50th percentile frequency is 30.
- The 75th percentile (third quartile) of item frequencies is 112.5 indicates that 75% of the unique items have a frequency of 112.5 or less.
- The maximum frequency among the unique items is 1545 which represents the most frequently purchased item in the dataset.

```
# Summary statistics for the frequency of unique item
df['Item'].value_counts().describe()
```

```
count     163.000000
mean      118.717791
std       226.130180
min         1.000000
25%         8.000000
50%        30.000000
75%       112.500000
max      1545.000000
Name: Item, dtype: float64
```

- Overview of dataset

The dataset includes 163 unique products, 3873 unique members who made transactions and records of transactions that took place over a span of 728 different days.

```
print(f"Number of product: {df['Item'].nunique()} products")
print(f"Number of member: {df['Member_id'].nunique()} members")
print(f"Number of transactions day: {df['Date'].nunique()} days")
```

```
Number of product: 163 products
Number of member: 3873 members
Number of transactions day: 728 days
```

- Daily item purchased

This shows daily sales patterns of the store that could be valuable for business planning. The maximum and minimum values give an idea of the range of transactions, with some days having very high sales up to 60 items and others having lower sales as low as 9 items. The average daily item sold is approximately 27 The average daily item sold helps to understand the typical sales volume per day across the entire dataset, serving as a reference point for daily sales expectations.

```
# Count of items purchased on each day
daily_item = df.groupby('Date').agg({'Item':'count'}).reset_index()
```

```
print(f"Maximum daily item sold: {daily_item['Item'].max()}")
print(f"Minimum daily item sold: {daily_item['Item'].min()}")
print(f"Average daily item sold: {round(daily_item['Item'].mean())}")
```

```
Maximum daily item sold: 60
Minimum daily item sold: 9
Average daily item sold: 27
```

- List of items for each transaction

The result indicates that there are a total of 13,971 transactions or baskets in the dataset. Over the period, customers made 13,971 distinct purchases and each transaction or basket contains a list of items that were bought on a specific date by a particular member.

```
# List of items for each transaction
df_baskets = df.groupby(['Date', 'Member_id'])['Item'].apply(list).reset_index()
```

```
df_baskets
```

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

13971 rows × 3 columns

```
print(f"Total transactions/baskets: {df_baskets.shape[0]}")
```

```
Total transactions/baskets: 13971
```

- Total transactions

In the year 2014, there were a total of 6,989 transactions recorded and 6,982 transactions in the year 2015.

```
# Create a new column to store the year for each transaction
df_baskets['Year'] = df_baskets['Date'].dt.year

# Group the data by year and count the number of unique baskets
total_transactions_by_year = df_baskets.groupby('Year')['Date'].count()
```

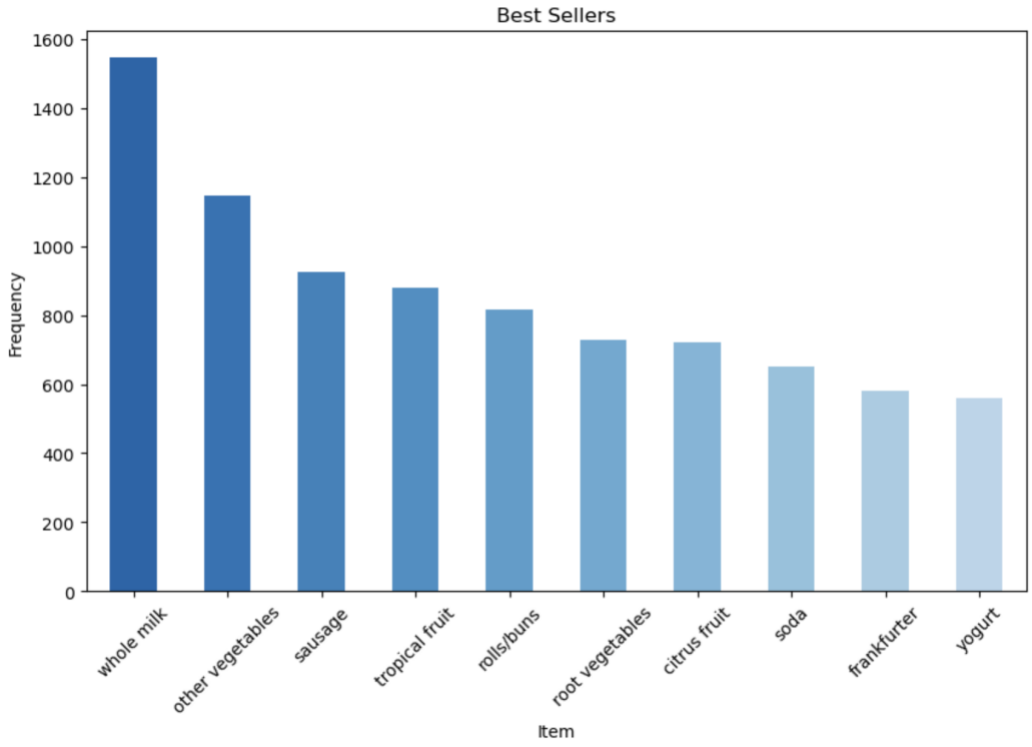
```
# Total transactions for each year
print(f"Total transactions in 2014: {total_transactions_by_year.get(2014, 0)}")
print(f"Total transactions in 2015: {total_transactions_by_year.get(2015, 0)}")
```

```
Total transactions in 2014: 6989
Total transactions in 2015: 6982
```

- Bar plot of the top 10 best-selling items

The bar plot visualise the top 10 best-selling items from the store during those two years, consists of whole milk, other vegetables, sausage, tropical fruit, rolls/buns, root vegetables, citrus fruit, soda, frankfurter and yogurt. By acknowledging this information, store owner can optimise the inventory by ensuring the top-selling items are always in stock, reducing the risk of stockouts and potential lost sales.

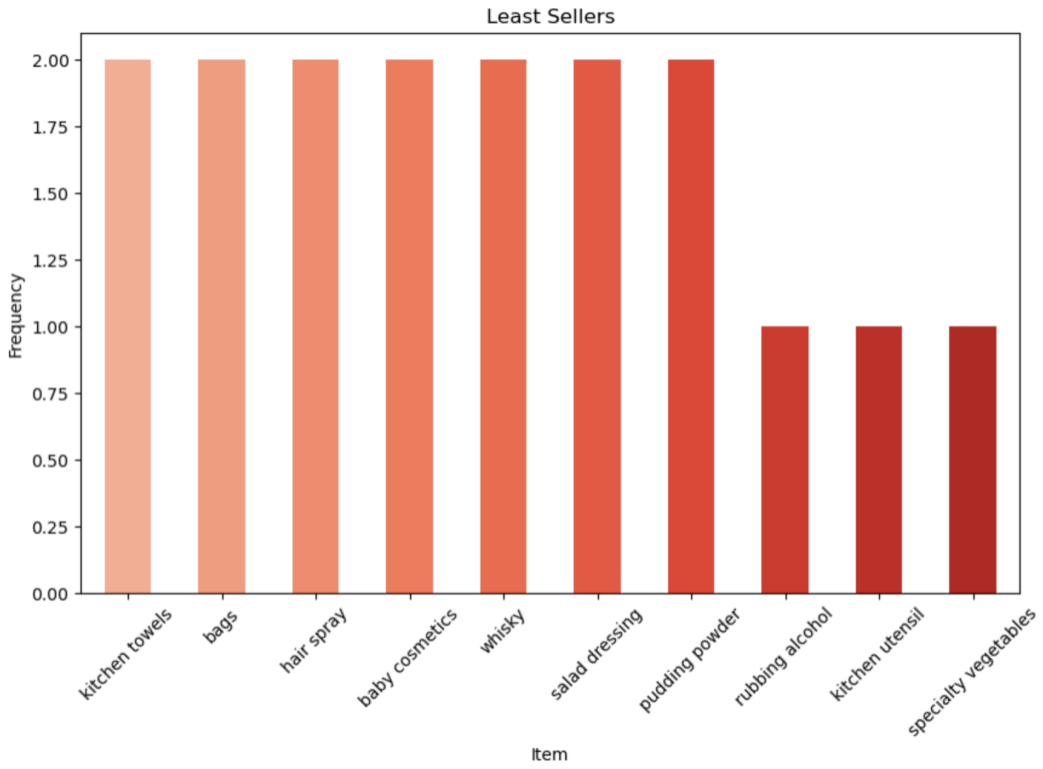
```
# Create a bar plot of the top 10 best-selling items
plt.figure(figsize=(10, 6))
df['Item'].value_counts().head(10).plot(kind='bar', color=plt.cm.Blues(np.linspace(0.8, 0.3, 10)))
plt.title("Best Sellers")
plt.xlabel("Item")
plt.ylabel("Frequency")
plt.xticks(rotation=45)
plt.show()
```




- Bar plot of the last 10 least-selling items

The bar plot visualise the top 10 least-selling items from the store, namely kitchen towels, bags, hair spray, baby cosmetics, whisky, salad dressing, pudding powder, rubbing alcohol, kitchen utensil and specialty vegetables. Identifying the least selling items in a store could help store owner to minimise the risk of overstocking and potential losses due to unsold products. At the same time, the owner can also experiment with targeted marketing or promotional efforts to boost sales of the existing least selling items or bundle them with complementary products.

```
# Create a bar plot of the 10 least-selling items
plt.figure(figsize=(10, 6))
df['Item'].value_counts().tail(10).plot(kind='bar', color=plt.cm.Reds(np.linspace(0.3, 0.8, 10)))
plt.title("Least Sellers")
plt.xlabel("Item")
plt.ylabel("Frequency")
plt.xticks(rotation=45)
plt.show()
```




- Yearly item sold

In the year 2014, a total of 7,000 items were sold. This means that customers purchased a combined total of 7,000 individual items in the year 2014. In the year 2015, a significantly larger number of items were sold with total of 12,351 items. This indicates that customers made a substantial number of individual item purchases in this year.

```
# Count of items purchased on each year
yearly_itemsold = df.groupby('Year')['Item'].agg(['count']).reset_index()
```

```
print(f"Total item sold in 2014: {yearly_itemsold[yearly_itemsold['Year'] == 2014]['count'].values[0]}")
print(f"Total item sold in 2015: {yearly_itemsold[yearly_itemsold['Year'] == 2015]['count'].values[0]}")
```

```
Total item sold in 2014: 7000
Total item sold in 2015: 12351
```

- Distribution of yearly item sold

Result indicates ‘whole milk’ consistently appeared as a popular item in both years, suggesting its ongoing popularity. The rankings of other common items changed slightly between the two years, but the general categories of items remained similar. There were some

unique and less common items that appeared in both years but had limited occurrences like ‘kitchen towels’.

```
# Count of item occurrences in the year 2014
df[df['Year'] == 2014]['Item'].value_counts()
```

```
whole milk          563
other vegetables    488
root vegetables     334
rolls/buns          327
tropical fruit      320
                   ...
organic products      1
frozen fruits         1
kitchen utensil       1
liqueur               1
kitchen towels        1
Name: Item, Length: 154, dtype: int64
```

```
# Count of item occurrences in the year 2015
df[df['Year'] == 2015]['Item'].value_counts()
```

```
whole milk          982
sausage             744
other vegetables    658
tropical fruit      559
frankfurter         505
                   ...
skin care             1
rice                  1
kitchen towels        1
salad dressing        1
nut snack             1
Name: Item, Length: 161, dtype: int64
```

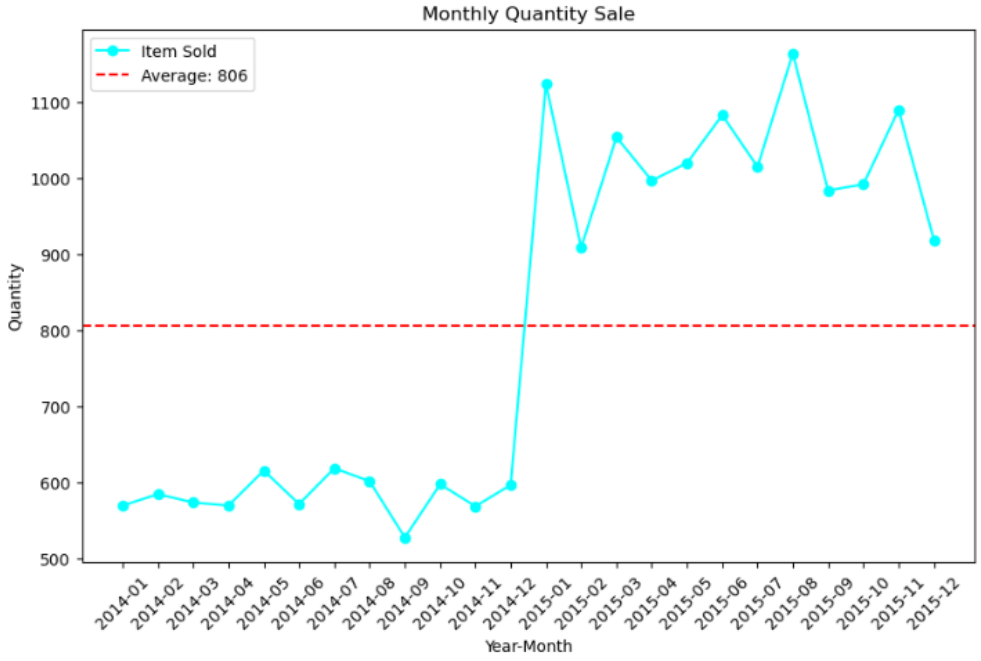
- Line plot of monthly quantity sale

The graphical depiction of monthly quantity sale exhibits significant variations in the quantity of goods sold across those two years. There is considerable rise in quantity item sold observed between December 2014 and January 2015, suggesting a sharp increase in client transactions during this period. In contrast, the month of September 2014 exhibits the lowest quantity sales figures. Furthermore, the visual analysis provides evidence that monthly quantity sale in the year of 2015 goes beyond average of monthly item sold. August 2015 exhibits the greatest quantity sales figures which indicating a notable surge in client engagement and transactions during that particular period. These information offers store owner significant insights regarding the sales of items, enabling a deeper comprehension of seasonal patterns and potential influences on client buying behaviour.

```python
# Group the data by 'YearMonth' and calculate the monthly quantity sale
df['YearMonth'] = df['Date'].dt.to_period('M')
average_monthly_quantitysale = df.groupby('YearMonth')['Item'].count().mean()

# Create a line plot for average monthly item sold
monthly_quantitysale = df.groupby('YearMonth')['Item'].count()
monthly_quantitysale.index = monthly_quantitysale.index.strftime('%Y-%m')  # Convert 'YearMonth' to strings

plt.figure(figsize=(10, 6))
plt.plot(monthly_quantitysale, marker='o', linestyle='-', color='aqua', label='Item Sold')
plt.axhline(average_monthly_quantitysale, color='r', linestyle='--', label=f'Average: {int(average_monthly_quantitysale)}')
plt.title('Monthly Quantity Sale')
plt.xlabel('Year-Month')
plt.ylabel('Quantity')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



**Preparation of Dataset**

- Perform transactions encoding

Group the data into individual baskets for each member, where a basket is a collection of items purchased by a member on a particular date.

```
# Create a basket of items purchased by each member
basket = (df.groupby(['Date', 'Member_id', 'Item'])['Item']
          .count().unstack().reset_index().drop('Date',axis=1).fillna(0)
          .set_index('Member_id'))
```

To prepare the data for association rule mining, encode the item counts as binary values, where 1 indicates that the item was purchased by the member and 0 indicates that it is not. The .applymap() function is used to apply a lambda function to each element in the dataframe. The lambda function checks if the count of an item is greater than 0 which means the item was purchased and assigns 1 if true or 0 if false.

```
# Encode item counts as binary (0 or 1)
df_encoded = basket.applymap(lambda x: 1 if x > 0 else 0)
```

```
df_encoded
```

| Item | 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 | wh bre |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Member_id | | | | | | | | | | | | | | | | | |
| 1249 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 1381 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 1440 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 1659 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 1789 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 3738 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 3971 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 4058 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 4565 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |
| 4863 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | |

13971 rows × 163 columns

**Methodology**

- Import necessary libraries for association rule mining

Import necessary functions from the mlxtend.frequent_patterns module for association rule mining. Apriori function is used to perform the Apriori algorithm, which is for finding frequent itemsets in transaction data. While association_rules function is used to generate association rules from the frequent itemsets obtained using the Apriori algorithm.

```
# Import necessary libraries
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

- Justification of the choice of minimum support

Generally, support of an itemset or an association rule is calculated using the following formula:

$$\text{Support (x)} = \frac{\text{(Number of transactions containing itemset x)}}{\text{(Total number of transactions)}}$$

To determine minimum support, least occurrence of item is divided by total baskets or transactions which resulting to 0.00007. This approach ensures that the Apriori algorithm considers all items, even those with the smallest support.

```
# Determine min_support
print(f'Minimum occurrences: {df.Item.value_counts().min()}')
print(f'Total baskets: {len(df_baskets)}')

df['Item'].value_counts().min()/len(df_baskets)
```

```
Minimum occurrences: 1
Total baskets: 13971

7.157683773530885e-05
```

- Applies Apriori algorithm

Applies the Apriori algorithm to discover frequent itemsets in the dataset represented by 'df_encoded' with minimum support parameter of 0.00007. Unfortunately, 0.00007 is considered as a small minimum support threshold as it will generate high number of frequent itemsets, which in turn leads to a high number of rules. This can result in misleading rules and might overshadow meaningful patterns.

```
# Find frequent itemsets
freq_itemsets = apriori(df_encoded, min_support = 0.00007, use_colnames = True)
```

```
freq_itemsets.sort_values(by = 'support', ascending = False)
```

| | support | itemsets |
|---|---|---|
| 160 | 0.108153 | (whole milk) |
| 101 | 0.080739 | (other vegetables) |
| 127 | 0.064634 | (sausage) |
| 152 | 0.061986 | (tropical fruit) |
| 119 | 0.057834 | (rolls/buns) |
| ... | ... | ... |
| 2289 | 0.000072 | (UHT-milk, chewing gum, sausage) |
| 2290 | 0.000072 | (UHT-milk, fish, chicken) |
| 2291 | 0.000072 | (UHT-milk, chicken, yogurt) |
| 2292 | 0.000072 | (UHT-milk, citrus fruit, chocolate) |
| 4574 | 0.000072 | (beverages, soft cheese, sausage, curd, specia... |

4575 rows × 2 columns

It often involves a trial-and-error process to find the right value for minimum support. Hence, the minimum support is then set to 0.00008. This choice of minimum support values seems to be optimal as it best serves the analysis in striking the right balance to extract meaningful and relevant associations.

The itemset ‘whole milk’ has a support of 0.108153, indicating that ‘whole milk’ appears in approximately 10.82% of the transactions. While 0.000143 is noticed as the lowest value of support for those particular itemsets.

```
# Find frequent itemsets
freq_itemsets = apriori(df_encoded, min_support = 0.00008, use_colnames = True)
```

```
freq_itemsets.sort_values(by = 'support', ascending = False)
```

| | support | itemsets |
|---|---|---|
| 157 | 0.108153 | (whole milk) |
| 100 | 0.080739 | (other vegetables) |
| 125 | 0.064634 | (sausage) |
| 149 | 0.061986 | (tropical fruit) |
| 118 | 0.057834 | (rolls/buns) |
| ... | ... | ... |
| 484 | 0.000143 | (chewing gum, other vegetables) |
| 900 | 0.000143 | (root vegetables, margarine) |
| 485 | 0.000143 | (photo/film, chewing gum) |
| 897 | 0.000143 | (margarine, pastry) |
| 1289 | 0.000143 | (pastry, whole milk, sausage, yogurt) |

1290 rows × 2 columns

- Justification of the choice of minimum threshold

Threshold determines the level of significance or the minimum level of support that an itemset or association rule must meet to be considered for inclusion in the results. To strike a

balance, minimum threshold value of 0.000358 from 75% quartile support for itemsets with a length of 2 is being choose. This choice is based on the observation that the majority of baskets in the dataset contain 2 items. If the threshold is set too low (e.g., 25% or 50% quartile), it would result in a large number of rules, potentially including many weak or trivial rules. If the threshold is set too high (e.g., at the maximum), it would result in very few rules, which may not capture important associations in the data. By setting the threshold at the 75% quartile, it may result to itemsets that are moderately frequent but not overly common, leading to a balance between the quantity and quality of generated rules.

```
# Determine min_threshold
freq_itemsets['length'] = freq_itemsets['itemsets'].apply(len)

freq_itemsets.groupby('length')['support'].describe()
```

| length | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| 1 | 160.0 | 0.008568 | 0.016023 | 0.000143 | 0.000644 | 0.002183 | 0.008267 | 0.108153 |
| 2 | 999.0 | 0.000386 | 0.000466 | 0.000143 | 0.000143 | 0.000215 | 0.000358 | 0.005225 |
| 3 | 130.0 | 0.000166 | 0.000047 | 0.000143 | 0.000143 | 0.000143 | 0.000143 | 0.000358 |
| 4 | 1.0 | 0.000143 | NaN | 0.000143 | 0.000143 | 0.000143 | 0.000143 | 0.000143 |

- Generate association rule

Generates association rules using the Apriori algorithm and the 'support' metric with a minimum threshold of 0.000358. Association rules generated from transaction data provide valuable insights into frequent item pairs. These rules reveal which items tend to be purchased together, facilitating product recommendations and optimised product placement. This help business owner identifies emerging trends, perform A/B testing and enhance customer satisfaction, ultimately leading to increased sales and improved customer experiences. Further interpretation of the generated association rules is included in Section 4.

```
# Generate association rules
rules = association_rules(freq_itemsets, metric="support", min_threshold=0.000358)
```

```
rules
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (UHT-milk) | (tropical fruit) | 0.011238 | 0.061986 | 0.000429 | 0.038217 | 0.616540 | -0.000267 | 0.975287 | -0.386135 |
| 1 | (tropical fruit) | (UHT-milk) | 0.061986 | 0.011238 | 0.000429 | 0.006928 | 0.616540 | -0.000267 | 0.995661 | -0.398697 |
| 2 | (UHT-milk) | (whole milk) | 0.011238 | 0.108153 | 0.000716 | 0.063694 | 0.588930 | -0.000500 | 0.952517 | -0.413809 |
| 3 | (whole milk) | (UHT-milk) | 0.108153 | 0.011238 | 0.000716 | 0.006618 | 0.588930 | -0.000500 | 0.995350 | -0.439034 |
| 4 | (beef) | (bottled beer) | 0.033570 | 0.024050 | 0.000787 | 0.023454 | 0.975232 | -0.000020 | 0.999390 | -0.025606 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 493 | (whole milk) | (whipped/sour cream) | 0.108153 | 0.015818 | 0.000930 | 0.008604 | 0.543894 | -0.000780 | 0.992722 | -0.484613 |
| 494 | (white bread) | (whole milk) | 0.007516 | 0.108153 | 0.000501 | 0.066667 | 0.616413 | -0.000312 | 0.955551 | -0.385372 |
| 495 | (whole milk) | (white bread) | 0.108153 | 0.007516 | 0.000501 | 0.004633 | 0.616413 | -0.000312 | 0.997104 | -0.410986 |
| 496 | (whole milk) | (yogurt) | 0.108153 | 0.039510 | 0.002505 | 0.023163 | 0.586262 | -0.001768 | 0.983265 | -0.441747 |
| 497 | (yogurt) | (whole milk) | 0.039510 | 0.108153 | 0.002505 | 0.063406 | 0.586262 | -0.001768 | 0.952224 | -0.423549 |

498 rows × 10 columns

Trial-and-error with different minimum thresholds is important in generating association rules because it allows us to fine-tune the rules to meet specific objectives. The choice of minimum threshold directly impacts the number of rules produced. Minimum threshold of 0.000215 from 50% quartile support for itemsets with a length of 2 is then being test on generating association rules. Unfortunately, 0.000215 seems to be not suitable as it lead to a high number of association rules, which may include noise or less meaningful associations. Thus, 0.000358 is the optimal value for minimum threshold as it yields a relevant number of rules. By experimenting with different thresholds, we can balance the comprehensiveness of the rules against their relevance to uncover the sweet spot.

```
# Generate association rules
rules = association_rules(freq_itemsets, metric="support", min_threshold=0.000215)
rules
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | (UHT-milk) | (other vegetables) | 0.011238 | 0.080739 | 0.000286 | 0.025478 | 0.315558 | -0.000621 | 0.943294 | -0.686878 |
| **1** | (other vegetables) | (UHT-milk) | 0.080739 | 0.011238 | 0.000286 | 0.003546 | 0.315558 | -0.000621 | 0.992281 | -0.702336 |
| **2** | (UHT-milk) | (pip fruit) | 0.011238 | 0.039582 | 0.000286 | 0.025478 | 0.643669 | -0.000158 | 0.985527 | -0.358927 |
| **3** | (pip fruit) | (UHT-milk) | 0.039582 | 0.011238 | 0.000286 | 0.007233 | 0.643669 | -0.000158 | 0.995967 | -0.365647 |
| **4** | (UHT-milk) | (pork) | 0.011238 | 0.037005 | 0.000286 | 0.025478 | 0.688489 | -0.000130 | 0.988171 | -0.313939 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **909** | (whole milk, yogurt) | (sausage) | 0.002505 | 0.064634 | 0.000358 | 0.142857 | 2.210252 | 0.000196 | 1.091260 | 0.548938 |
| **910** | (sausage, yogurt) | (whole milk) | 0.001718 | 0.108153 | 0.000358 | 0.208333 | 1.926291 | 0.000172 | 1.126544 | 0.481695 |
| **911** | (whole milk) | (sausage, yogurt) | 0.108153 | 0.001718 | 0.000358 | 0.003309 | 1.926291 | 0.000172 | 1.001597 | 0.539181 |
| **912** | (sausage) | (whole milk, yogurt) | 0.064634 | 0.002505 | 0.000358 | 0.005537 | 2.210252 | 0.000196 | 1.003049 | 0.585399 |
| **913** | (yogurt) | (whole milk, sausage) | 0.039510 | 0.005225 | 0.000358 | 0.009058 | 1.733547 | 0.000151 | 1.003868 | 0.440554 |

914 rows × 10 columns

**Result and Interpretation**

- Interpretation of the generated association rules

Zhang's metric is a measure designed to assess the strength of association either positive or negative between two items, taking into account both their co-occurrence and their non-co-occurrence. It is particularly useful to understand how the presence or absence of an item affects the likelihood of another item being present in a transaction. It has a value between -1 and 1, with positive values representing association and negative values representing disassociation. Value close to zero indicates no significant association.

While high confidence indicates that, for the selected rules, the consequent items are frequently purchased when the antecedent items are purchased. In other words, there is a high probability that the rules predictions are correct.

There are 33 association rules included in specific threshold of 0.5 for Zhang's metric with the filtered rules are sort in descending order of confidence. The rules are identified as strong as they have high confidence and a strong Zhang's metric. These rules are likely to be meaningful and actionable in a business context.

Moreover, all lift values from the included generated rules are observed to be greater than 1. Whereby lift value that is greater than 1 indicates a positive association between items. This means that the presence of one item in the rules is positively correlated with the presence of another item.

The store owner can leverage from these filtered association rules for various purposes. The owner can optimise the store layout by placing frequently associated items close to each other. For example, 'pastry', 'sausage' and 'whole milk' are often purchased together, hence they can be strategically positioned to boost sales. Besides, the owner can design effective promotion strategies by offering discounts or bundling products that are frequently bought together as this may increase sales and customer satisfaction. Moreover, association rules can guide cross-selling strategies. For instance, if customers frequently buy 'pasta', offering 'sausage' can increase the average transaction value.

```
# Filter and sort rules based on specific conditions
rules[rules['zhangs_metric'] > 0.5].sort_values(by='confidence', ascending=False)
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| **891** | (pastry, sausage) | (whole milk) | 0.001074 | 0.108153 | 0.000358 | 0.333333 | 3.082065 | 0.000242 | 1.337771 | 0.676268 |
| **890** | (pastry, whole milk) | (sausage) | 0.001718 | 0.064634 | 0.000358 | 0.208333 | 3.223283 | 0.000247 | 1.181515 | 0.690944 |
| **867** | (frankfurter, sausage) | (other vegetables) | 0.001575 | 0.080739 | 0.000286 | 0.181818 | 2.251934 | 0.000159 | 1.123542 | 0.556814 |
| **909** | (whole milk, yogurt) | (sausage) | 0.002505 | 0.064634 | 0.000358 | 0.142857 | 2.210252 | 0.000196 | 1.091260 | 0.548938 |
| **514** | (frozen fish) | (tropical fruit) | 0.002648 | 0.061986 | 0.000358 | 0.135135 | 2.180107 | 0.000194 | 1.084579 | 0.542744 |
| **714** | (pasta) | (sausage) | 0.002720 | 0.064634 | 0.000358 | 0.131579 | 2.035758 | 0.000182 | 1.077088 | 0.510170 |
| **300** | (hygiene articles) | (chicken) | 0.003364 | 0.028702 | 0.000358 | 0.106383 | 3.706425 | 0.000261 | 1.086928 | 0.732663 |
| **868** | (other vegetables, sausage) | (frankfurter) | 0.002935 | 0.040441 | 0.000286 | 0.097561 | 2.412433 | 0.000168 | 1.063295 | 0.587204 |
| **873** | (other vegetables, whole milk) | (frankfurter) | 0.004080 | 0.040441 | 0.000358 | 0.087719 | 2.169073 | 0.000193 | 1.051824 | 0.541182 |
| **41** | (long life bakery product) | (beef) | 0.004939 | 0.033570 | 0.000429 | 0.086957 | 2.590340 | 0.000264 | 1.058471 | 0.616998 |
| **322** | (waffles) | (chicken) | 0.005010 | 0.028702 | 0.000429 | 0.085714 | 2.986320 | 0.000286 | 1.062357 | 0.668489 |
| **892** | (whole milk, sausage) | (pastry) | 0.005225 | 0.021473 | 0.000358 | 0.068493 | 3.189726 | 0.000246 | 1.050477 | 0.690099 |
| **550** | (grapes) | (onions) | 0.010164 | 0.012884 | 0.000358 | 0.035211 | 2.732981 | 0.000227 | 1.023142 | 0.640610 |
| **215** | (coffee) | (butter) | 0.012598 | 0.015604 | 0.000429 | 0.034091 | 2.184789 | 0.000233 | 1.019140 | 0.549209 |
| **74** | (butter milk) | (berries) | 0.008446 | 0.016248 | 0.000286 | 0.033898 | 2.086314 | 0.000149 | 1.018270 | 0.525121 |
| **551** | (onions) | (grapes) | 0.012884 | 0.010164 | 0.000358 | 0.027778 | 2.732981 | 0.000227 | 1.018117 | 0.642375 |
| **214** | (butter) | (coffee) | 0.015604 | 0.012598 | 0.000429 | 0.027523 | 2.184789 | 0.000233 | 1.015348 | 0.550886 |
| **75** | (berries) | (butter milk) | 0.016248 | 0.008446 | 0.000286 | 0.017621 | 2.086314 | 0.000149 | 1.009340 | 0.529286 |
| **893** | (pastry) | (whole milk, sausage) | 0.021473 | 0.005225 | 0.000358 | 0.016667 | 3.189726 | 0.000246 | 1.011635 | 0.701558 |
| **323** | (chicken) | (waffles) | 0.028702 | 0.005010 | 0.000429 | 0.014963 | 2.986320 | 0.000286 | 1.010103 | 0.684795 |

This condition filters the rules based on their Zhang's metric values. It selects rules with a Zhang's metric less than -0.5, which implies that the rules have a relatively strong negative association according to this metric. Negative association rules indicate that when the antecedent items are purchased, the consequent items are less likely to be purchased.

Store owner can gain valuable insights from this filtered association rules for more effective marketing strategies. In this case, when two or more items have a strong negative association, it indicates that customers tend to avoid purchasing them together. Store owners can use this information to identify products that are not compatible with each other, possibly due to differences in taste, use or customer preferences. Distinct customer segments with varying preferences help store owner to tailor marketing and promotional strategies to specific customer groups. Besides, negative associations can also provide feedback on product quality, taste or packaging. Store owners may improve those products or packaging to increase their appeal to customers. Furthermore, store layout and product placement can be optimised based on negative associations whereby items that are frequently avoided together can be placed apart on the shelves to encourage exploration and increase sales.

```
# Filter rules based on specific condition
rules[rules['zhangs_metric'] < -0.5]
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | (UHT-milk) | (other vegetables) | 0.011238 | 0.080739 | 0.000286 | 0.025478 | 0.315558 | -0.000621 | 0.943294 | -0.686878 |
| **1** | (other vegetables) | (UHT-milk) | 0.080739 | 0.011238 | 0.000286 | 0.003546 | 0.315558 | -0.000621 | 0.992281 | -0.702336 |
| **6** | (UHT-milk) | (sausage) | 0.011238 | 0.064634 | 0.000286 | 0.025478 | 0.394185 | -0.000440 | 0.959820 | -0.608511 |
| **7** | (sausage) | (UHT-milk) | 0.064634 | 0.011238 | 0.000286 | 0.004430 | 0.394185 | -0.000440 | 0.993162 | -0.621653 |
| **16** | (beef) | (bottled water) | 0.033570 | 0.026698 | 0.000358 | 0.010661 | 0.399315 | -0.000538 | 0.983790 | -0.608846 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **849** | (tropical fruit) | (whipped/sour cream) | 0.061986 | 0.015818 | 0.000358 | 0.005774 | 0.364995 | -0.000623 | 0.989897 | -0.649704 |
| **852** | (whole milk) | (tropical fruit) | 0.108153 | 0.061986 | 0.003364 | 0.031105 | 0.501814 | -0.003340 | 0.968128 | -0.526775 |
| **853** | (tropical fruit) | (whole milk) | 0.061986 | 0.108153 | 0.003364 | 0.054273 | 0.501814 | -0.003340 | 0.943028 | -0.514179 |
| **856** | (turkey) | (whole milk) | 0.005297 | 0.108153 | 0.000286 | 0.054054 | 0.499794 | -0.000287 | 0.942810 | -0.501533 |
| **857** | (whole milk) | (turkey) | 0.108153 | 0.005297 | 0.000286 | 0.002647 | 0.499794 | -0.000287 | 0.997344 | -0.528789 |

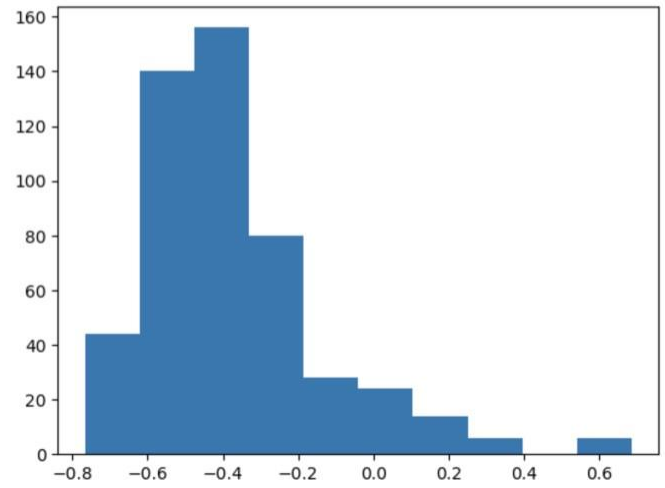
265 rows × 10 columns

- Histogram of Zhang's metric

The histogram plot of Zhang's metric appears to be right-skewed indicates that there are relatively fewer association rules with high values of Zhang's metric. This means that majority of the rules have lower values of the metric.

```
plt.hist(rules['zhangs_metric'])
```

```
(array([ 44., 140., 156.,  80.,  28.,  24.,  14.,   6.,   0.,   6.]),
 array([-0.76604853, -0.62096419, -0.47587984, -0.3307955 , -0.18571116,
        -0.04062682,  0.10445752,  0.24954187,  0.39462621,  0.53971055,
         0.68479489]),
 <BarContainer object of 10 artists>)
```

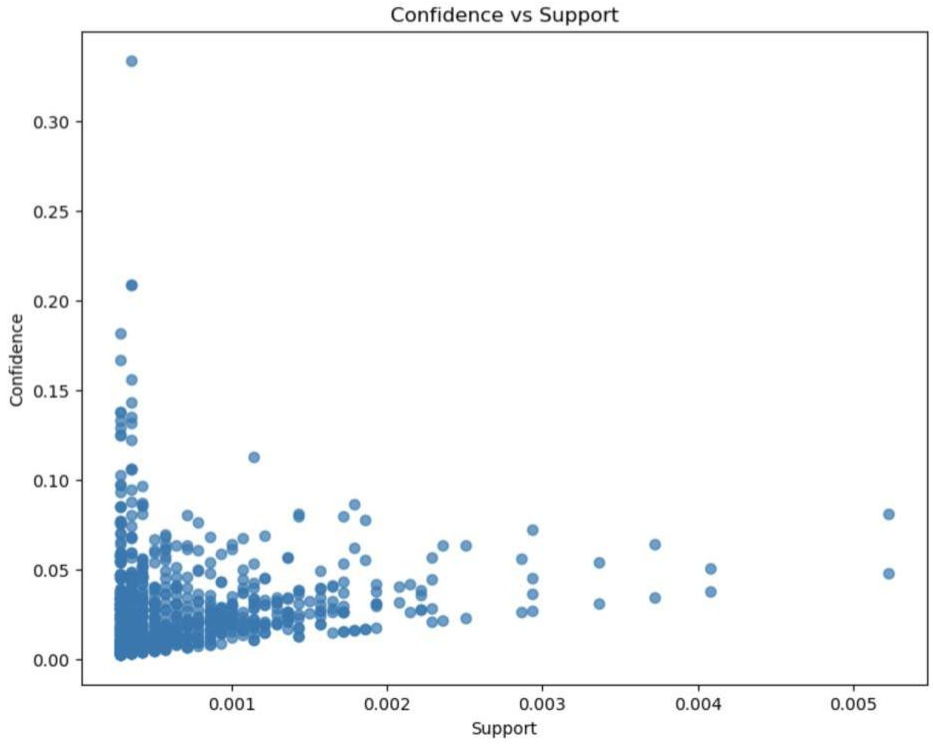



- Scatter plot of confidence vs support

The graph indicates scatter plot of confidence vs support for the generated association rules. Confidence on the y-axis measures the probability of the consequent being purchased when the antecedent is purchased. Higher confidence values indicate stronger associations between items. While support on the x-axis indicates the frequency of occurrence of the itemset in the dataset. Support measures how often the itemset appears in the transactions.

There is presence of a cluster of points and forming a positive linear relationship. This suggests that there is a correlation between the support and confidence of the association rules. As support increases, confidence tends to increase as well. This indicates that items with higher support are more likely to have higher confidence, implying more reliable associations between those items.

```
# Create a scatter plot of confidence vs. support for association rules
plt.figure(figsize=(8, 6))
plt.scatter(rules['support'], rules['confidence'], alpha=0.7)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title("Confidence vs Support for Association Rules")
plt.show()
```




- Scatter plot of lift vs support

The graph indicates scatter plot of lift vs support for the generated association rules. Lift is represented on the y-axis indicates the strength of the association between items. Lift measures how much more likely items are purchased together than if they were purchased independently. A lift value greater than 1 suggests a positive association, while a lift less than 1 suggests a negative association. Support on the x-axis indicates the frequency of occurrence of the itemset in the dataset. Support measures how often the itemset appears in the transactions.

There is cluster or groups of points in the scatter plot which represent different groups of association rules with quite similar support and lift values. Points with positive association

which are values greater than 1 seems to have low support value whereby they did not happen very frequently.

```
# Create a scatter plot of lift vs. support for association rules
plt.figure(figsize=(8, 6))
plt.scatter(rules['support'], rules['lift'], alpha=0.7)
plt.xlabel('Support')
plt.ylabel('Lift')
plt.title("Lift vs Support for Association Rules")
plt.show()
```