Ee Szen

In [1]:
```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
import seaborn as sns

# Load the datasets
products = pd.read_csv('data/products.csv')
transactions = pd.read_csv('data/transactions.csv')
stores = pd.read_csv('data/stores.csv')
```

In [2]:
```
products
```

Out[2]:

	category	avg_price	margin	shelf_life_days
0	Fresh Produce	4.99	0.35	7
1	Dairy	3.99	0.25	14
2	Meat	12.99	0.30	5
3	Bakery	3.99	0.45	3
4	Beverages	3.49	0.50	180
5	Snacks	2.99	0.40	90
6	Canned Goods	2.49	0.35	365
7	Frozen Foods	5.99	0.40	180
8	Household	6.99	0.45	365
9	Personal Care	7.99	0.50	365

In [3]:
```
transactions
```

Out[3]:

	transaction_id	store_id	date	product_id	category	quantity	unit_price	i
0	T000001	13	2023-07-26 17:41:00	HOU_006	Household	2	6.75	
1	T000001	13	2023-07-26 17:41:00	PER_028	Personal Care	1	8.59	
2	T000001	13	2023-07-26 17:41:00	BEV_001	Beverages	1	3.59	
3	T000001	13	2023-07-26 17:41:00	FRE_039	Fresh Produce	1	5.12	
4	T000001	13	2023-07-26 17:41:00	FRO_037	Frozen Foods	3	5.70	
...	...	...	...	...	...	...	...	
850543	T199999	2	2023-03-06 21:21:00	MEA_020	Meat	2	13.16	
850544	T199999	2	2023-03-06 21:21:00	FRO_032	Frozen Foods	2	6.38	
850545	T200000	35	2023-03-11 13:14:00	FRE_045	Fresh Produce	4	4.84	
850546	T200000	35	2023-03-11 13:14:00	MEA_025	Meat	2	14.32	
850547	T200000	35	2023-03-11 13:14:00	FRO_033	Frozen Foods	2	6.85	

850548 rows × 9 columns

In [4]:
```
stores
```

Out[4]:

	store_id	location_type	store_size	region
0	1	Rural	Medium	West
1	2	Urban	Small	West
2	3	Rural	Medium	West
3	4	Rural	Small	East
4	5	Urban	Medium	East
5	6	Urban	Large	East
6	7	Rural	Large	North
7	8	Suburban	Small	West
8	9	Rural	Large	East
9	10	Rural	Large	East
10	11	Rural	Medium	North
11	12	Rural	Small	East
12	13	Urban	Medium	North
13	14	Rural	Medium	South
14	15	Suburban	Medium	East
15	16	Urban	Medium	South
16	17	Suburban	Medium	North
17	18	Suburban	Medium	West
18	19	Suburban	Medium	East
19	20	Suburban	Small	North
20	21	Urban	Large	West
21	22	Urban	Medium	West
22	23	Suburban	Medium	South
23	24	Suburban	Medium	North
24	25	Urban	Medium	West
25	26	Urban	Medium	East
26	27	Urban	Medium	East
27	28	Rural	Large	South
28	29	Rural	Large	West
29	30	Rural	Medium	North

	store_id	location_type	store_size	region
30	31	Suburban	Large	East
31	32	Rural	Small	West
32	33	Suburban	Medium	West
33	34	Suburban	Small	South
34	35	Rural	Small	East
35	36	Suburban	Medium	East
36	37	Rural	Large	North
37	38	Rural	Small	East
38	39	Urban	Medium	North
39	40	Rural	Small	East
40	41	Urban	Small	South
41	42	Rural	Small	East
42	43	Rural	Small	North
43	44	Urban	Large	North
44	45	Urban	Small	South
45	46	Rural	Small	East
46	47	Suburban	Small	East
47	48	Urban	Large	South
48	49	Suburban	Small	East
49	50	Suburban	Small	East

In [5]:
```
# Data Cleaning & Preprocessing
transactions['date'] = pd.to_datetime(transactions['date'])  # Convert date column
transactions['date']
```

Out[5]:
```
0         2023-07-26 17:41:00
1         2023-07-26 17:41:00
2         2023-07-26 17:41:00
3         2023-07-26 17:41:00
4         2023-07-26 17:41:00
                  ...
850543    2023-03-06 21:21:00
850544    2023-03-06 21:21:00
850545    2023-03-11 13:14:00
850546    2023-03-11 13:14:00
850547    2023-03-11 13:14:00
Name: date, Length: 850548, dtype: datetime64[ns]
```

In [6]:
```
products.fillna('Unknown', inplace=True)
stores.fillna('Unknown', inplace=True)
transactions.fillna(0, inplace=True)
```

In [7]:
```
# Join datasets for enrichment
# Feature Engineering
transactions['total_price'] = transactions['quantity'] * transactions['unit_price']
transactions['day_of_week'] = transactions['date'].dt.day_name()  # Extract day of
transactions['hour_of_day'] = transactions['date'].dt.hour  # Extract hour of the d

transactions = transactions.merge(products, left_on='category', right_on='category'
transactions = transactions.merge(stores, on='store_id', how='left')
transactions
```

Out[7]:

	transaction_id	store_id	date	product_id	category	quantity	unit_price	i
0	T000001	13	2023-07-26 17:41:00	HOU_006	Household	2	6.75	
1	T000001	13	2023-07-26 17:41:00	PER_028	Personal Care	1	8.59	
2	T000001	13	2023-07-26 17:41:00	BEV_001	Beverages	1	3.59	
3	T000001	13	2023-07-26 17:41:00	FRE_039	Fresh Produce	1	5.12	
4	T000001	13	2023-07-26 17:41:00	FRO_037	Frozen Foods	3	5.70	
...	...	...	...	...	...	...	...	
850543	T199999	2	2023-03-06 21:21:00	MEA_020	Meat	2	13.16	
850544	T199999	2	2023-03-06 21:21:00	FRO_032	Frozen Foods	2	6.38	
850545	T200000	35	2023-03-11 13:14:00	FRE_045	Fresh Produce	4	4.84	
850546	T200000	35	2023-03-11 13:14:00	MEA_025	Meat	2	14.32	
850547	T200000	35	2023-03-11 13:14:00	FRO_033	Frozen Foods	2	6.85	

850548 rows × 18 columns

In [8]:
```
store_metrics = transactions.groupby('store_id').agg(
    total_revenue=('total_price', 'sum'),
    unique_transactions=('transaction_id', 'nunique'),
    total_items=('quantity', 'sum')
).reset_index()
```

In [9]:
```
store_metrics['average_transaction_value'] = store_metrics['total_revenue'] / store
store_metrics['average_transaction_value']
```

Out[9]:
```
0     52.377632
1     51.974940
2     51.825400
3     52.727496
4     52.339768
5     53.040351
6     52.218202
7     52.713711
8     51.962730
9     53.510670
10    51.555228
11    52.487658
12    52.769842
13    52.445161
14    51.749257
15    52.860555
16    51.825878
17    52.186995
18    53.169786
19    53.015056
20    51.323772
21    52.875345
22    52.347923
23    51.801408
24    51.367341
25    52.175328
26    52.134988
27    52.177488
28    53.640171
29    51.463679
30    52.971798
31    52.880860
32    52.703490
33    52.423628
34    51.547519
35    52.548411
36    52.299080
37    52.176124
38    52.931100
39    53.716631
40    53.607133
41    52.668215
42    52.120336
43    52.163856
44    52.509110
45    52.728087
46    52.620438
47    52.346769
48    51.560442
49    52.992591
Name: average_transaction_value, dtype: float64
```

In [10]:
```
# Save processed data
transactions.to_csv('data/processed_transactions.csv', index=False)
store_metrics.to_csv('data/store_metrics.csv', index=False)
```

In [11]:
```
# Display summary for validation
print("Data cleaned and processed. Summary:")
store_metrics.head()
```

```
Data cleaned and processed. Summary:
```

Out[11]:

	store_id	total_revenue	unique_transactions	total_items	average_transaction_value
0	1	206158.36	3936	36959	52.377632
1	2	202546.34	3897	36341	51.974940
2	3	206731.52	3989	37163	51.825400
3	4	212280.90	4026	37791	52.727496
4	5	205485.93	3926	36881	52.339768

In [12]:
```
# Save enriched transactions for further analysis
transactions.head()
```

Out[12]:

	transaction_id	store_id	date	product_id	category	quantity	unit_price	is_loya
0	T000001	13	2023-07-26 17:41:00	HOU_006	Household	2	6.75	
1	T000001	13	2023-07-26 17:41:00	PER_028	Personal Care	1	8.59	
2	T000001	13	2023-07-26 17:41:00	BEV_001	Beverages	1	3.59	
3	T000001	13	2023-07-26 17:41:00	FRE_039	Fresh Produce	1	5.12	
4	T000001	13	2023-07-26 17:41:00	FRO_037	Frozen Foods	3	5.70	

In [13]:
```
# Association Rule Mining
basket = (transactions.groupby(['transaction_id', 'product_id'])
          ['quantity'].sum().unstack().fillna(0).map(lambda x: 1 if x > 0 else 0))
```

Logan

In [14]:
```
frequent_itemsets = apriori(basket, min_support=0.05, use_colnames=True, low_memory
rules = association_rules(frequent_itemsets, num_itemsets=len(frequent_itemsets), m
rules = rules.sort_values('lift', ascending=False)
```

```
C:\Users\Logan Kannan\.anaconda\Lib\site-packages\mlxtend\frequent_patterns\fpcommo
n.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computa
tionalperformance and their support might be discontinued in the future.Please use a
DataFrame with bool type
  warnings.warn(
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[14], line 2
      1 frequent_itemsets = apriori(basket, min_support=0.05, use_colnames=True, low
_memory=True)
----> 2 rules = association_rules(frequent_itemsets, num_itemsets=len(frequent_items
ets), metric="lift", min_threshold=1.0)
      3 rules = rules.sort_values('lift', ascending=False)

File ~\.anaconda\Lib\site-packages\mlxtend\frequent_patterns\association_rules.py:12
6, in association_rules(df, num_itemsets, df_orig, null_values, metric, min_threshol
d, support_only, return_metrics)
    123 fpc.valid_input_check(df_orig, null_values)
    125 if not df.shape[0]:
--> 126     raise ValueError(
    127         "The input DataFrame `df` containing " "the frequent itemsets is emp
ty."
    128     )
    130 # check for mandatory columns
    131 if not all(col in df.columns for col in ["support", "itemsets"]):

ValueError: The input DataFrame `df` containing the frequent itemsets is empty.
```

In [ ]:
```
rules.to_csv('data/association_rules.csv', index=False)
```

In [ ]:
```
high_freq_categories = frequent_itemsets.sort_values('support', ascending=False).he
print("High-frequency product categories:", high_freq_categories.tolist())
```

In [ ]:
```
seasonal_promo = transactions.groupby(transactions['date'].dt.month)['total_price']
promo_calendar = seasonal_promo.reset_index()
promo_calendar.columns = ['Month', 'Revenue']
```

In [ ]:
```
promo_calendar.to_csv('data/promo_calendar.csv', index=False)
```

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='Month', y='Revenue', data=promo_calendar)
plt.title('Seasonal Revenue Trends')
plt.xlabel('Month')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('figures/seasonal_trends.png')
```

In [ ]:
```
current_average_transaction_value = 45
new_average_transaction_value = store_metrics['average_transaction_value'].mean()
increase_percentage = ((new_average_transaction_value - current_average_transaction
```

In [ ]:
```
print(f"Current Average Transaction Value: ${current_average_transaction_value}")
print(f"New Average Transaction Value: ${new_average_transaction_value:.2f}")
```

```
print(f"Increase in Transaction Value: {increase_percentage:.2f}%")
```