Association Rule Mining

Import Required Packages

In [1]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

Load the Dataset

In [2]:
```
# Load the dataset
df = pd.read_csv('bank.csv')

# Display the first few rows of the dataset
df.head()
```

Out[2]:

	Account No	DATE	TRANSACTION DETAILS	CHQ.NO.	VALUE DATE	WITHDRAWAL AMT	DEPOSIT AMT	BALANCE AMT	.
0	409000611074'	29-Jun-17	TRF FROM Indiaforensic SERVICES	NaN	29-Jun-17	NaN	1,000,000.00	1,000,000.00	.
1	409000611074'	05-Jul-17	TRF FROM Indiaforensic SERVICES	NaN	05-Jul-17	NaN	1,000,000.00	2,000,000.00	.
2	409000611074'	18-Jul-17	FDRLINTERNAL FUND TRANSFE	NaN	18-Jul-17	NaN	500,000.00	2,500,000.00	.
3	409000611074'	01-Aug-17	TRF FRM Indiaforensic SERVICES	NaN	01-Aug-17	NaN	3,000,000.00	5,500,000.00	.
4	409000611074'	16-Aug-17	FDRLINTERNAL FUND TRANSFE	NaN	16-Aug-17	NaN	500,000.00	6,000,000.00	.

Prepare the Data

In [4]:
```
df.columns = df.columns.str.strip()
print(df.columns)
```

```
Index(['Account No', 'DATE', 'TRANSACTION DETAILS', 'CHQ.NO.', 'VALUE DATE',
       'WITHDRAWAL AMT', 'DEPOSIT AMT', 'BALANCE AMT', '.'],
      dtype='object')
```

In [5]:
```
# Combine relevant columns into a single string per transaction
df['transaction'] = df.apply(lambda row: f"{row['TRANSACTION DETAILS']},{row['WITHDRAWAL AMT']},{row['DEPOSIT AMT']}", axis=1)

# Create a list of transactions
transactions = df['transaction'].apply(lambda x: x.split(','))

# Convert to list of lists
data = transactions.tolist()
```

Transform Data using Transaction Encoder

In [6]:
```
# Initialize TransactionEncoder
te = TransactionEncoder()

# Transform the data
te_data = te.fit(data).transform(data)

# Convert to DataFrame
df_transformed = pd.DataFrame(te_data, columns=te.columns_)
df_transformed.head()
```

Out[6]:

	Indiaforensic SERVICE	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	...	ZEN LEFIN PVT LTD16120746	ZEN LEFIN PVT LTD161223B0	ZEN LEFIN PVT LTD161223B3D	ZEN LEFIN PVT LTD1701098B	ZEN LEFIN PVT LTD1701213CF	ZEN LEFIN PVT LTD170125A0	ZEN LEFIN PVT LTD1701302E	ZEN LEFIN PVT LTD170206BA	Z LTD
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	

5 rows × 64476 columns

Apply Apriori Algorithm

In [7]:
```
# Apply apriori algorithm
frequent_itemsets = apriori(df_transformed, min_support=0.05, use_colnames=True)

# Add a length column for the number of items in the itemset
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

Out[7]:

	support	itemsets	length
0	0.090679	(1)	1
1	0.063115	(10)	1
2	0.083467	(15)	1
3	0.053494	(2)	1
4	0.218759	(000)	1
5	0.509058	(000.00)	1
6	0.076066	(FDRLINTERNAL FUND TRANSFE)	1
7	0.053889	(FDRLINATIONAL ELECTRONIC F)	1
8	1.000000	(nan)	1
9	0.090679	(nan, 1)	2
10	0.058055	(000.00 , 10)	2
11	0.063115	(nan, 10)	2
12	0.076084	(000, 15)	2
13	0.079380	(15, 000.00)	2
14	0.083467	(15, nan)	2
15	0.053494	(2, nan)	2
16	0.218527	(000, 000.00)	2
17	0.218759	(000, nan)	2
18	0.075344	(000.00 , FDRLINTERNAL FUND TRANSFE)	2
19	0.509058	(nan, 000.00)	2
20	0.076066	(nan, FDRLINTERNAL FUND TRANSFE)	2
21	0.053889	(nan, FDRLINATIONAL ELECTRONIC F)	2
22	0.058055	(nan, 000.00 , 10)	3
23	0.076084	(000, 15, 000.00)	3
24	0.076084	(15, 000, nan)	3
25	0.079380	(15, nan, 000.00)	3
26	0.218527	(000, nan, 000.00)	3
27	0.075344	(nan, 000.00 , FDRLINTERNAL FUND TRANSFE)	3
28	0.076084	(15, 000, nan, 000.00)	4

Filter the Frequent Itemsets

In [8]:
```
# Filter for itemsets of length >= 2 and support >= 0.05
filtered_itemsets = frequent_itemsets[(frequent_itemsets['length'] >= 2) & (frequent_itemsets['support'] >= 0.05)]
filtered_itemsets
```

Out[8]:

	support	itemsets	length
9	0.090679	(nan, 1)	2
10	0.058055	(000.00 , 10)	2
11	0.063115	(nan, 10)	2
12	0.076084	(000, 15)	2
13	0.079380	(15, 000.00)	2
14	0.083467	(15, nan)	2
15	0.053494	(2, nan)	2
16	0.218527	(000, 000.00)	2
17	0.218759	(000, nan)	2
18	0.075344	(000.00 , FDRLINTERNAL FUND TRANSFE)	2
19	0.509058	(nan, 000.00)	2
20	0.076066	(nan, FDRLINTERNAL FUND TRANSFE)	2
21	0.053889	(nan, FDRLINATIONAL ELECTRONIC F)	2
22	0.058055	(nan, 000.00 , 10)	3
23	0.076084	(000, 15, 000.00)	3
24	0.076084	(15, 000, nan)	3
25	0.079380	(15, nan, 000.00)	3
26	0.218527	(000, nan, 000.00)	3
27	0.075344	(nan, 000.00 , FDRLINTERNAL FUND TRANSFE)	3
28	0.076084	(15, 000, nan, 000.00)	4

Generate and Display Association Rules

In [9]:
```
# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Add length columns for antecedents and consequents
rules['antecedents length'] = rules['antecedents'].apply(lambda x: len(x))
rules['consequents length'] = rules['consequents'].apply(lambda x: len(x))

# Display rules
rules
```

Out[9]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric	antecedents length	consequents length
0	(nan)	(1)	1.000000	0.090679	0.090679	0.090679	1.000000	0.000000	1.000000	0.000000	1	1
1	(1)	(nan)	0.090679	1.000000	0.090679	1.000000	1.000000	0.000000	inf	0.000000	1	1
2	(000.00)	(10)	0.509058	0.063115	0.058055	0.114043	1.806918	0.025926	1.057484	0.909621	1	1
3	(10)	(000.00)	0.063115	0.509058	0.058055	0.919825	1.806918	0.025926	6.123421	0.476656	1	1
4	(nan)	(10)	1.000000	0.063115	0.063115	0.063115	1.000000	0.000000	1.000000	0.000000	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
71	(nan, 000.00)	(000, 15)	0.509058	0.076084	0.076084	0.149460	1.964414	0.037353	1.086270	1.000000	2	2
72	(15)	(000, nan, 000.00)	0.083467	0.218527	0.076084	0.911537	4.171289	0.057844	8.833929	0.829502	1	3
73	(000)	(15, 000.00 , nan)	0.218759	0.079380	0.076084	0.347797	4.381435	0.058719	1.411555	0.987870	1	3
74	(nan)	(000, 15, 000.00)	1.000000	0.076084	0.076084	0.076084	1.000000	0.000000	1.000000	0.000000	1	3
75	(000.00)	(000, 15, nan)	0.509058	0.076084	0.076084	0.149460	1.964414	0.037353	1.086270	1.000000	1	3

76 rows × 12 columns

Filter and Display the Rules

In [10]:
```
# Filter rules based on antecedents length and lift
filtered_rules = rules[(rules['antecedents length'] > 1) & (rules['consequents length'] == 1) & (rules['lift'] > 1)]

# Sort and display the rules by confidence and lift
filtered_rules_sorted = filtered_rules.sort_values(by=["confidence", "lift"], ascending=False)

# Display relevant columns
filtered_rules_sorted[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
```

Out[10]:

	antecedents	consequents	support	confidence	lift
32	(000, 15)	(000.00)	0.076084	1.000000	1.964414
62	(000, 15, nan)	(000.00)	0.076084	1.000000	1.964414
50	(000, nan)	(000.00)	0.218527	0.998938	1.962328
57	(nan, FDRLINTERNAL FUND TRANSFE)	(000.00)	0.075344	0.990497	1.945746
34	(15, 000.00)	(000)	0.076084	0.958478	4.381435
64	(15, 000.00 , nan)	(000)	0.076084	0.958478	4.381435
44	(15, nan)	(000.00)	0.079380	0.951026	1.868209
27	(nan, 10)	(000.00)	0.058055	0.919825	1.806918
39	(15, nan)	(000)	0.076084	0.911537	4.166858
52	(nan, 000.00)	(000)	0.218527	0.429277	1.962328
33	(000, 000.00)	(15)	0.076084	0.348167	4.171289
65	(000, nan, 000.00)	(15)	0.076084	0.348167	4.171289
40	(000, nan)	(15)	0.076084	0.347797	4.166858
46	(nan, 000.00)	(15)	0.079380	0.155935	1.868209
56	(nan, 000.00) (FDRLINTERNAL FUND TRANSFE)		0.075344	0.148006	1.945746
26	(nan, 000.00)	(10)	0.058055	0.114043	1.806918

Learning Reflection on Association Rule Mining (ARM) Implementation

Implementing Association Rule Mining (ARM) in Python using the Apriori algorithm provided a valuable learning experience, especially in the context of analyzing a bank transaction dataset. Here are the key reflections from this process:

Understanding the dataset

- Initial Exploration: The initial steps involved loading and exploring the dataset. Understanding the structure and content of the dataset was crucial. The dataset contained columns like 'Account No', 'DATE', 'TRANSACTION DETAILS', 'WITHDRAWAL AMT', 'DEPOSIT AMT', and 'BALANCE AMT'. This step highlighted the importance of data familiarity before applying any mining techniques.
- Data Preparation: Preparing the data by combining relevant columns into a single string per transaction was a critical step. This transformation made it possible to analyze the data in a meaningful way for ARM. The use of a lambda function to concatenate transaction details with withdrawal and deposit amounts demonstrated the need for creative data manipulation to fit the algorithm's requirements.

Transforming the data

TransactionEncoder: The use of TransactionEncoder from the mlxtend library was an enlightening experience. It transformed the list of transactions into a format suitable for the Apriori algorithm, specifically a one-hot encoded DataFrame. This transformation was essential for identifying the presence of items in each transaction efficiently.

Applying the Apriori Algorithm

- Apriori Algorithm: Applying the Apriori algorithm to the transformed data was a core part of the process. This algorithm helped identify frequent itemsets based on a minimum support threshold. It was interesting to see how changing the support threshold could affect the number and type of itemsets generated.
- Frequent Itemsets: The algorithm's ability to discover frequent itemsets provided insight into the common combinations of transaction details and amounts in the dataset. Adding a length column to indicate the number of items in each itemset helped in further analysis and filtering.

Generating Association Rules

- Association Rules: Generating association rules from the frequent itemsets was a crucial step. It involved setting a minimum lift threshold to ensure that the rules identified were significant. The rules provided actionable insights into how certain transaction details were related to each other.
- Filtering and Sorting Rules: Filtering rules based on antecedent and consequent lengths, and sorting by confidence and lift, highlighted the most interesting and useful patterns. This step reinforced the importance of not just generating rules but also evaluating and interpreting them effectively.

Conclusion

The implementation of Association Rule Mining using the Apriori algorithm provided a comprehensive learning experience. It demonstrated the importance of data preparation, the intricacies of applying machine learning algorithms, and the significance of interpreting results to derive actionable insights. This project not only deepened my understanding of ARM but also showcased its practical applications in the real world, particularly in the financial sector.