STUDENT NAME: ODUFUWA ODUTAYO THOMSON

MATRIC NO: 209074209

COURSE CODE: MIT 816

FREQUENT ITEMSET MINING IN PYTHON

TASK 4 - Frequent Itemset Mining in Python

Import the necessary libraries

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

Loading the dataset

```
dataset = [['Milk', 'Onion', 'Bread', 'Kidney Beans', 'Eggs', 'Yoghurt'],
           ['Fish', 'Onion', 'Bread', 'Kidney Beans', 'Eggs', 'Yoghurt'],
           ['Milk', 'Apples', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Sugar', 'Tea Leaves', 'Kidney Beans', 'Yoghurt'],
           ['Tea Leaves', 'Onion', 'Kidney Beans', 'Ice Cream', 'Eggs']]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Transforming the data

```
tr_encoder = TransactionEncoder()
tr_array = tr_encoder.fit(dataset).transform(dataset)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Converting to a pandas dataframe

```
df = pd.DataFrame(tr_array, columns=tr_encoder.columns_)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
df
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
```

APPLYING THE APRIORI ALGORITHM

```
frequent_itemsets_ap = apriori(df, min_support=0.6, use_colnames = True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
frequent_itemsets_ap
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yoghurt)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Onion, Eggs)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Yoghurt, Kidney Beans)
10	0.6	(Onion, Kidney Beans, Eggs)

APPLYING THE FP GROWTH ALGORITHM

```
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
frequent_itemsets_fp = fpgrowth(df, min_support=0.6, use_colnames=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
frequent_itemsets_fp
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
    support          itemsets
```

```
rules_apriori = association_rules(frequent_itemsets_ap, metric="lift", min_threshold=1.0)
rules_fpgrowth = association_rules(frequent_itemsets_fp, metric="lift", min_threshold=1.0)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
rules_apriori
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.0
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.0
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.1
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.1
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.0
5	(Kidney Beans)	(Milk)	1.0	0.6	0.6	0.60	1.00	0.0
6	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.0
7	(Kidney Beans)	(Onion)	1.0	0.6	0.6	0.60	1.00	0.0
8	(Yoghurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.0
9	(Kidney Beans)	(Yoghurt)	1.0	0.6	0.6	0.60	1.00	0.0
10	(Onion, Kidney	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.1

```
rules_fpgrowth
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Comparing both results

```
0   (Kidney Beans)   (Eggs)   1.0   0.8   0.8   0.80   1.00   0.00   1.0   0.0
```

A comparison of the two algorithms:

1. Execution Time:

> Apriori: Apriori uses a level-wise approach, where it generates candidate itemsets of different lengths and scans the database multiple times. As a result, it can be slower on large datasets with many items.

> FP-growth: FP-growth is generally faster than Apriori because it uses a divide-and-conquer strategy. It builds a frequent pattern tree (FP-tree) from the dataset and then recursively mines frequent itemsets from the tree structure. This reduces the number of database scans, making it more efficient.

2. Memory Usage:

> Apriori: Apriori requires additional memory to store candidate itemsets and support counts for each level of itemsets. This memory usage can become significant for large datasets with high item cardinality.

> FP-growth: FP-growth is memory-efficient because it constructs the FP-tree and uses a compact data structure. It usually requires less memory compared to Apriori.

3. Scalability:

> Apriori: Apriori may struggle to scale to large datasets due to its multiple database scans and memory requirements. It can be impractical for very large datasets.

> FP-growth: FP-growth is more scalable and can handle large datasets more effectively. It is often the preferred choice for mining frequent itemsets in big data scenarios.

4. Minimum Support Threshold:

> Apriori: You need to specify a minimum support threshold in Apriori to control the selection of frequent itemsets. This threshold can affect the number of itemsets generated.

> FP-growth: Similarly, you set a minimum support threshold in FP-growth. Adjusting this threshold will impact the number and size of frequent itemsets. Association Rule Generation:

In summary, FP-growth is generally preferred for its speed and memory efficiency, especially on large datasets. Apriori can still be suitable for smaller datasets or when you need more control over the itemset generation process.

```
import timeit
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

The code below would be used to practically evaluate both algorithms

```
# Measure execution time
def measure_execution_time(frequent_itemsets):
    start_time = timeit.default_timer()
    frequent_itemsets
    execution_time = timeit.default_timer() - start_time
    return execution_time

# Memory usage (approximate)
def measure_memory_usage(frequent_itemsets):
    import sys
    return sys.getsizeof(frequent_itemsets)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
# Measure execution time
apriori_execution_time = measure_execution_time(frequent_itemsets_ap)
print(f"Apriori execution time is: {apriori_execution_time}")

# Measure memory usage
apriori_memory_usage = measure_memory_usage(frequent_itemsets_ap)
print(f"Apriori memory usage is: {apriori_memory_usage}")
```

```
Apriori execution time is: 3.5099992601317354e-07
Apriori memory usage is: 2520
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
# Measure execution time
fpgrowth_execution_time = measure_execution_time(frequent_itemsets_fp)
print(f"FP Growth execution time is: {fpgrowth_execution_time}")

# Measure memory usage
fpgrowth_memory_usage = measure_memory_usage(frequent_itemsets_fp)
print(f"FP Growth memory usage is: {fpgrowth_memory_usage}")
```

```
FP Growth execution time is: 3.000000106112566e-07
FP Growth memory usage is: 2520
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
def better_alg():
  if apriori_execution_time >= fpgrowth_execution_time or apriori_memory_usage >= fpgrowth_memory_usage:
    print("FP Growth Algorithm is faster and it also takes less memory space compared to Apriori Algorithm")
  else:
    print("Apriori Algorithm is faster and it also takes less memory space compared to FP Growth Algorithm")

better_alg()
```

```
FP Growth Algorithm is faster and it also takes less memory space compared to Apriori Algorithm
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

FP Growth runs faster than