SET A) :1)

```
# Dataset initialization
transactions = {
    1: ['Bread', 'Milk'],
    2: ['Bread', 'Diaper', 'Beer', 'Eggs'],
    3: ['Milk', 'Diaper', 'Beer', 'Coke'],
    4: ['Bread', 'Milk', 'Diaper', 'Beer'],
    5: ['Bread', 'Milk', 'Diaper', 'Coke']
}

# Step 1: Get the unique items in the dataset
unique_items = sorted({item for transaction in transactions.values()
for item in transaction})

# Step 2: Create a dictionary to map items to numeric values
item_to_numeric = {item: idx + 1 for idx, item in
enumerate(unique_items)}

# Step 3: Convert transactions to numeric format (binary matrix)
numeric_transactions = []
for tid, items in transactions.items():
    row = [1 if item in items else 0 for item in unique_items]
    numeric_transactions.append(row)

# Step 4: Display the results
print("Unique Items:", unique_items)
print("\nItem to Numeric Mapping:", item_to_numeric)
print("\nBinary Matrix Representation:")
print("TID", "\t".join(unique_items))
for i, row in enumerate(numeric_transactions, start=1):
    print(f"{i}\t" + "\t".join(map(str, row)))
```

```
Unique Items: ['Beer', 'Bread', 'Coke', 'Diaper', 'Eggs', 'Milk']

Item to Numeric Mapping: {'Beer': 1, 'Bread': 2, 'Coke': 3, 'Diaper':
4, 'Eggs': 5, 'Milk': 6}

Binary Matrix Representation:
TID Beer   Bread Coke  Diaper     Eggs  Milk
1    0     1     0     0     0     1
2    1     1     0     1     1     0
3    1     0     1     1     0     1
4    1     1     0     1     0     1
5    0     1     1     1     0     1
```

2)

```
# Custom Transactions Dataset
transactions = {
    1: ['Pen', 'Notebook', 'Eraser'],
```

```
    2: ['Notebook', 'Ruler', 'Pen', 'Marker'],
    3: ['Eraser', 'Ruler', 'Sharpener'],
    4: ['Pen', 'Notebook', 'Sharpener'],
    5: ['Notebook', 'Pen', 'Marker', 'Eraser']
}

# Step 1: Get the unique items in the dataset
unique_items = sorted({item for transaction in transactions.values()
for item in transaction})

# Step 2: Create a dictionary to map items to numeric values
item_to_numeric = {item: idx + 1 for idx, item in
enumerate(unique_items)}

# Step 3: Convert transactions to numeric format (binary matrix)
numeric_transactions = []
for tid, items in transactions.items():
    row = [1 if item in items else 0 for item in unique_items]
    numeric_transactions.append(row)

# Step 4: Display the results
print("Unique Items:", unique_items)
print("\nItem to Numeric Mapping:", item_to_numeric)
print("\nBinary Matrix Representation:")
print("TID", "\t".join(unique_items))
for i, row in enumerate(numeric_transactions, start=1):
    print(f"{i}\t" + "\t".join(map(str, row)))
```

```
Unique Items: ['Eraser', 'Marker', 'Notebook', 'Pen', 'Ruler',
'Sharpener']

Item to Numeric Mapping: {'Eraser': 1, 'Marker': 2, 'Notebook': 3,
'Pen': 4, 'Ruler': 5, 'Sharpener': 6}

Binary Matrix Representation:
TID Eraser Marker     Notebook   Pen   Ruler Sharpener
1     1     0     1     1     0     0
2     0     1     1     1     1     0
3     1     0     0     0     1     1
4     0     0     1     1     0     1
5     1     1     1     1     0     0
```

```
SET B):1)

# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Step 1: Load the dataset
# Replace 'groceries.csv' with the path to your dataset
file_path = "C:\\Users\\ecs\\OneDrive\\Videos\\Documents\\Desktop\\
```

```
dataset\\Market_Basket_Optimisation.csv"
data = pd.read_csv(file_path)

# Step 2: Display dataset information
print("Dataset Information:")
print(data.info())
print("\nFirst 5 Rows of the Dataset:")
print(data.head())

# Step 3: Preprocess the data
# Dropping null values
data = data.dropna()
print("\nAfter Dropping Null Values:")
print(data.info())

# Step 4: Convert categorical values to numeric format
# One-hot encoding for items
print("\nConverting categorical values to one-hot encoding...")
data['Items'] = data['Items'].str.split(',')  # Split items into a
list
unique_items = sorted({item for sublist in data['Items'] for item in
sublist})
encoded_data = pd.DataFrame(
    [{item: (item in transaction) for item in unique_items} for
transaction in data['Items']]
)

print("\nOne-hot Encoded Data (First 5 Rows):")
print(encoded_data.head())

# Step 5: Apply Apriori Algorithm
# Generate frequent itemsets with minimum support of 0.2 (changeable)
frequent_itemsets = apriori(encoded_data, min_support=0.2,
use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Step 6: Generate Association Rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)
print("\nAssociation Rules:")
print(rules)

# Save results to files (optional)
frequent_itemsets.to_csv("frequent_itemsets.csv", index=False)
rules.to_csv("association_rules.csv", index=False)
print("\nResults saved to 'frequent_itemsets.csv' and
'association_rules.csv'.")
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   shrimp             7500 non-null   object
 1   almonds            5746 non-null   object
 2   avocado            4388 non-null   object
 3   vegetables mix     3344 non-null   object
 4   green grapes       2528 non-null   object
 5   whole weat flour   1863 non-null   object
 6   yams               1368 non-null   object
 7   cottage cheese     980 non-null    object
 8   energy drink       653 non-null    object
 9   tomato juice       394 non-null    object
 10  low fat yogurt     255 non-null    object
 11  green tea          153 non-null    object
 12  honey              86 non-null     object
 13  salad              46 non-null     object
 14  mineral water      24 non-null     object
 15  salmon             7 non-null      object
 16  antioxydant juice  3 non-null      object
 17  frozen smoothie    3 non-null      object
 18  spinach            2 non-null      object
 19  olive oil          0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 1.1+ MB
None

First 5 Rows of the Dataset:
           shrimp    almonds     avocado    vegetables mix green
grapes  \
0         burgers  meatballs        eggs               NaN
NaN
1         chutney        NaN         NaN               NaN
NaN
2          turkey    avocado         NaN               NaN
NaN
3   mineral water       milk  energy bar  whole wheat rice    green
tea
4  low fat yogurt        NaN         NaN               NaN
NaN

  whole weat flour yams cottage cheese energy drink tomato juice  \
0              NaN  NaN            NaN          NaN          NaN
1              NaN  NaN            NaN          NaN          NaN
2              NaN  NaN            NaN          NaN          NaN
3              NaN  NaN            NaN          NaN          NaN
4              NaN  NaN            NaN          NaN          NaN
```

```
  low fat yogurt green tea honey salad mineral water salmon
antioxydant juice  \
0            NaN       NaN   NaN   NaN           NaN    NaN
NaN
1            NaN       NaN   NaN   NaN           NaN    NaN
NaN
2            NaN       NaN   NaN   NaN           NaN    NaN
NaN
3            NaN       NaN   NaN   NaN           NaN    NaN
NaN
4            NaN       NaN   NaN   NaN           NaN    NaN
NaN

  frozen smoothie spinach  olive oil
0             NaN     NaN        NaN
1             NaN     NaN        NaN
2             NaN     NaN        NaN
3             NaN     NaN        NaN
4             NaN     NaN        NaN

After Dropping Null Values:
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   shrimp             0 non-null      object
 1   almonds            0 non-null      object
 2   avocado            0 non-null      object
 3   vegetables mix     0 non-null      object
 4   green grapes       0 non-null      object
 5   whole weat flour   0 non-null      object
 6   yams               0 non-null      object
 7   cottage cheese     0 non-null      object
 8   energy drink       0 non-null      object
 9   tomato juice       0 non-null      object
 10  low fat yogurt     0 non-null      object
 11  green tea          0 non-null      object
 12  honey              0 non-null      object
 13  salad              0 non-null      object
 14  mineral water      0 non-null      object
 15  salmon             0 non-null      object
 16  antioxydant juice  0 non-null      object
 17  frozen smoothie    0 non-null      object
 18  spinach            0 non-null      object
 19  olive oil          0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 0.0+ bytes
None
```

```
Converting categorical values to one-hot encoding...

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call
last)
File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\indexes\base.py:3791, in Index.get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Items'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[5], line 25
     22 # Step 4: Convert categorical values to numeric format
     23 # One-hot encoding for items
     24 print("\nConverting categorical values to one-hot
encoding...")
---> 25 data['Items'] = data['Items'].str.split(',')  # Split items
into a list
     26 unique_items = sorted({item for sublist in data['Items'] for
item in sublist})
     27 encoded_data = pd.DataFrame(
     28     [{item: (item in transaction) for item in unique_items}
for transaction in data['Items']]
     29 )

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\frame.py:3893, in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]
```

```
File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\indexes\base.py:3798, in Index.get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will
raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'Items'
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("irfanasrullah/groceries")

print("Path to dataset files:", path)
```

```
Downloading from
https://www.kaggle.com/api/v1/datasets/download/irfanasrullah/grocerie
s?dataset_version_number=2...

100%|
                                                                    
          | 168k/168k [00:01<00:00, 162kB/s]

Extracting files...
Path to dataset files: C:\Users\ecs\.cache\kagglehub\datasets\
irfanasrullah\groceries\versions\2
```

```
2)

# Import necessary libraries
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Step 1: Load the dataset
file_path = "C:\\Users\\ecs\\OneDrive\\Videos\\Documents\\Desktop\\
dataset\\Market_Basket_Optimisation.csv"  # Update the path if needed
data = pd.read_csv(file_path)
```

```
# Step 2: Display dataset information
print("Dataset Information:")
print(data.info())
print("\nFirst 5 Rows of the Dataset:")
print(data.head())

# Step 3: Preprocess the data
# Dropping null values
data = data.dropna()
print("\nAfter Dropping Null Values:")
print(data.info())

# Step 4: Convert categorical data to numeric (One-Hot Encoding)
# Assuming 'items' column contains the list of items in each
transaction
data['items'] = data['items'].str.split(",")  # Split items into a
list
all_items = sorted({item for sublist in data['items'] for item in
sublist})  # Get unique items

# One-hot encoding
encoded_data = pd.DataFrame(
    [{item: (item in transaction) for item in all_items} for
transaction in data['items']]
)

print("\nOne-hot Encoded Data (First 5 Rows):")
print(encoded_data.head())

# Step 5: Apply Apriori Algorithm
# Generate frequent itemsets with a minimum support threshold
frequent_itemsets = apriori(encoded_data, min_support=0.02,
use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Step 6: Generate Association Rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)
print("\nAssociation Rules:")
print(rules)

# Step 7: Save results (optional)
frequent_itemsets.to_csv("frequent_itemsets.csv", index=False)
rules.to_csv("association_rules.csv", index=False)
print("\nResults saved to 'frequent_itemsets.csv' and
'association_rules.csv'.")
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7500 entries, 0 to 7499
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   shrimp             7500 non-null   object
 1   almonds            5746 non-null   object
 2   avocado            4388 non-null   object
 3   vegetables mix     3344 non-null   object
 4   green grapes       2528 non-null   object
 5   whole weat flour   1863 non-null   object
 6   yams               1368 non-null   object
 7   cottage cheese     980 non-null    object
 8   energy drink       653 non-null    object
 9   tomato juice       394 non-null    object
 10  low fat yogurt     255 non-null    object
 11  green tea          153 non-null    object
 12  honey              86 non-null     object
 13  salad              46 non-null     object
 14  mineral water      24 non-null     object
 15  salmon             7 non-null      object
 16  antioxydant juice  3 non-null      object
 17  frozen smoothie    3 non-null      object
 18  spinach            2 non-null      object
 19  olive oil          0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 1.1+ MB
None

First 5 Rows of the Dataset:
           shrimp    almonds     avocado    vegetables mix green
grapes  \
0         burgers  meatballs        eggs               NaN
NaN
1         chutney        NaN         NaN               NaN
NaN
2          turkey    avocado         NaN               NaN
NaN
3   mineral water       milk  energy bar  whole wheat rice    green
tea
4  low fat yogurt        NaN         NaN               NaN
NaN

  whole weat flour yams cottage cheese energy drink tomato juice  \
0              NaN  NaN            NaN          NaN          NaN
1              NaN  NaN            NaN          NaN          NaN
2              NaN  NaN            NaN          NaN          NaN
3              NaN  NaN            NaN          NaN          NaN
4              NaN  NaN            NaN          NaN          NaN

  low fat yogurt green tea honey salad mineral water salmon
```

```
antioxydant juice  \
0             NaN        NaN   NaN   NaN            NaN     NaN
NaN
1             NaN        NaN   NaN   NaN            NaN     NaN
NaN
2             NaN        NaN   NaN   NaN            NaN     NaN
NaN
3             NaN        NaN   NaN   NaN            NaN     NaN
NaN
4             NaN        NaN   NaN   NaN            NaN     NaN
NaN

  frozen smoothie spinach  olive oil
0              NaN     NaN        NaN
1              NaN     NaN        NaN
2              NaN     NaN        NaN
3              NaN     NaN        NaN
4              NaN     NaN        NaN

After Dropping Null Values:
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   shrimp              0 non-null      object
 1   almonds             0 non-null      object
 2   avocado             0 non-null      object
 3   vegetables mix      0 non-null      object
 4   green grapes        0 non-null      object
 5   whole weat flour    0 non-null      object
 6   yams                0 non-null      object
 7   cottage cheese      0 non-null      object
 8   energy drink        0 non-null      object
 9   tomato juice        0 non-null      object
 10  low fat yogurt      0 non-null      object
 11  green tea           0 non-null      object
 12  honey               0 non-null      object
 13  salad               0 non-null      object
 14  mineral water       0 non-null      object
 15  salmon              0 non-null      object
 16  antioxydant juice   0 non-null      object
 17  frozen smoothie     0 non-null      object
 18  spinach             0 non-null      object
 19  olive oil           0 non-null      float64
dtypes: float64(1), object(19)
memory usage: 0.0+ bytes
None
```

```
---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\indexes\base.py:3791, in Index.get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'items'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[7], line 23
     19 print(data.info())
     21 # Step 4: Convert categorical data to numeric (One-Hot
Encoding)
     22 # Assuming 'items' column contains the list of items in each
transaction
---> 23 data['items'] = data['items'].str.split(",")  # Split items
into a list
     24 all_items = sorted({item for sublist in data['items'] for item
in sublist})  # Get unique items
     26 # One-hot encoding

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\frame.py:3893, in DataFrame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\
pandas\core\indexes\base.py:3798, in Index.get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
```

```
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will
raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'items'
```

SET C)

```python
# Import necessary libraries
from itertools import combinations
from collections import defaultdict

# Function to calculate support of itemsets
def calculate_support(transactions, itemsets, min_support):
    itemset_counts = defaultdict(int)
    for transaction in transactions:
        for itemset in itemsets:
            if itemset.issubset(transaction):
                itemset_counts[itemset] += 1

    # Filter out itemsets below min_support
    num_transactions = len(transactions)
    frequent_itemsets = {itemset: support / num_transactions for
itemset, support in itemset_counts.items() if support /
num_transactions >= min_support}
    return frequent_itemsets

# Apriori Algorithm
def apriori(transactions, min_support=0.5):
    transactions = list(map(set, transactions))  # Convert
transactions to sets
    items = sorted({item for transaction in transactions for item in
transaction})  # Unique items
    single_itemsets = [frozenset([item]) for item in items]

    frequent_itemsets = {}
    k = 1
    current_itemsets = single_itemsets

    # Iteratively generate frequent itemsets of size k
    while current_itemsets:
        # Calculate support for current itemsets
        frequent_k_itemsets = calculate_support(transactions,
```

```
current_itemsets, min_support)
        frequent_itemsets.update(frequent_k_itemsets)

        # Generate new itemsets of size k+1
        current_itemsets = list(map(frozenset,
combinations(frequent_k_itemsets.keys(), k + 1)))
        k += 1

    return frequent_itemsets

# Generate Association Rules
def generate_association_rules(frequent_itemsets, min_confidence=0.5):
    rules = []
    for itemset, support in frequent_itemsets.items():
        if len(itemset) > 1:
            for antecedent in map(frozenset, combinations(itemset,
len(itemset) - 1)):
                consequent = itemset - antecedent
                confidence = support / frequent_itemsets[antecedent]
                if confidence >= min_confidence:
                    rules.append((antecedent, consequent, support,
confidence))
    return rules
```

# Test dataset

transactions = [ ["Milk", "Bread", "Beer"], ["Milk", "Diaper", "Beer", "Eggs"], ["Milk", "Bread", "Diaper", "Beer"], ["Bread", "Diaper", "Milk", "Eggs"], ["Bread", "Milk", "Diaper", "Beer", "Eggs"]]

# Parameters

min_support = 0.4 min_confidence = 0.6

# Apply Apriori Algorithm

frequent_itemsets = apriori(transactions, min_support=min_support) print("Frequent Itemsets:") for itemset, support in frequent_itemsets.items(): print(f"{set(itemset)}: {support:.2f}")

# Generate Association Rules

rules = generate_association_rules(frequent_itemsets, min_confidence=min_confidence) print("\nAssociation Rules:") for antecedent, consequent, support, confidence in rules:

```
print(f"{set(antecedent)} => {set(consequent)} (Support: {support:.2f}, Confidence: {confidence:.2f})")
```