Section 4
Market Basket Analysis
with Python

1. Dataset

Dataset

Market Basket Optimization

https://www.kaggle.com/roshansharma/market-basket-optimization/version/1

This dataset contains information about customers buying different grocery items at a Mall.

Load the Dataset

```
# Import Libraries
import pandas as pd
import numpy as np

# Load the Dataset
df = pd.read_csv("Market_Basket_Optimisation.csv", header=None)

# Show the shape of the data: the number of rows and columns
df.shape
(7501, 20)
```

```python
# Show the first five rows of the dataset
df.head()
```

	0	1	2	3	4	5	6	...
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	...
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	...
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	...
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	...
4	mineral water	milk	energy bar	Whole wheat rice	grean tea	NaN	NaN	...

2. Data Pre-processing

Data Preparation

```
# Create a list of transaction
df['Transactions']= df.values.tolist()
df['Transactions’]
```

```
0      [shrimp, almonds, avocado, vegetables mix, gre...
1      [burgers, meatballs, eggs, nan, nan, nan, nan,...
2      [chutney, nan, nan, nan, nan, nan, nan, nan, n...
3      [turkey, avocado, nan, nan, nan, nan, nan, nan...
...
7497   [burgers, frozen vegetables, eggs, french frie...
7498   [chicken, nan, nan, nan, nan, nan, nan, nan, n...
7499   [escalope, green tea, nan, nan, nan, nan, nan,...
7500   [eggs, frozen smoothie, yogurt cake, low fat y...
Name: Transactions, Length: 7501, dtype: object
```

```
# Delete NaN from the transaction list
df['Transactions'] = df['Transactions'].apply(lambda x: [i for i in x if str(i) != "nan"])
df['Transactions’]
```

```
0       [shrimp, almonds, avocado, vegetables mix, gre...
1                             [burgers, meatballs, eggs]
2                                              [chutney]
3                                      [turkey, avocado]
4       [mineral water, milk, energy bar, whole wheat ...
                              ...
7496                       [butter, light mayo, fresh bread]
7497    [burgers, frozen vegetables, eggs, french frie...
7498                                           [chicken]
7499                                [escalope, green tea]
7500    [eggs, frozen smoothie, yogurt cake, low fat y...
Name: Transactions, Length: 7501, dtype: object
```

```python
# Convert the transaction list from a DataFrame column into a list of string
transactions = list(df['Transactions’])


# Count a transaction which contains burgers, meatballs, and eggs
transactions.count(['burgers', 'meatballs', 'eggs’])
```

1

```python
# Import library to count the number of permutations
from itertools import permutations

# Extract unique items.
unique_items = [item for transaction in transactions for item in transaction]

# Convert the unique item list from a string to a list
unique_item_list = list(set(unique_items))

# Compute rules.
rules = list(permutations(unique_item_list, 2))

# Print the number of rules with length 2
print(len(rules))
```

```
14280
```

```python
# Import the library for encoding
from mlxtend.preprocessing import TransactionEncoder

# Instantiate transaction encoder
encoder = TransactionEncoder().fit(transactions)

# One-hot encode itemsets by applying transform
onehot = encoder.transform(transactions)

# Convert one-hot encoded data to DataFrame
onehot = pd.DataFrame(onehot, columns = encoder.columns_)
```

```
# Show the one-hot encoded dataframe
print(onehot)
```

	asparagus	almonds	antioxydant juice	asparagus	...
0	False	True	True	False	...
1	False	False	False	False	...
2	False	False	False	False	...
3	False	False	False	False	...
...	...	...	...	...	...
7497	False	False	False	False	...
7498	False	False	False	False	...
7499	False	False	False	False	...
7500	False	False	False	False	...

[7501 rows x 120 columns]

3. Basic Metrics

Support

```
# Computing Support for Single Items
print(onehot.mean())

asparagus            0.000133
almonds              0.020397
antioxydant juice    0.008932
                       ...
whole wheat rice     0.058526
yams                 0.011465
yogurt cake          0.027330
zucchini             0.009465
Length: 120, dtype: float64
```

$$Support(X) = \frac{Frequency(X)}{N}$$

```python
# Define itemset that contains both eggs and ground beef
onehot['eggs_&_ground beef'] = np.logical_and(onehot['eggs'], onehot['ground beef'])

# Compute Support for itemset that contains both eggs and ground beef
print(onehot['eggs_&_ground beef'].mean())
```

```
0.019997333688841486
```

```python
# Drop the column of "eggs_&_ground beef" to keep the dataset simple
onehot = onehot.drop('eggs_&_ground beef', axis=1)
```

Confidence

$$Confidence(X \rightarrow Y) = \frac{freq(X,\ \ Y)}{freq(X)}$$

$$= \frac{freq(X,Y)}{N} \cdot \frac{N}{freq(X)}$$

$$= \frac{Support(X\&Y)}{Support(X)}$$

Confidence (eggs → ground beef)

```
# Compute Support for the itemsets that contains eggs and/or ground beef
sup_eggs_groundbeef = np.logical_and(onehot['eggs'], onehot['ground beef']).mean()
sup_eggs = onehot['eggs'].mean()
sup_groundbeef = onehot['ground beef'].mean()

# Compute Confidence {eggs -> ground beef}
conf_eggs_to_groundbeef = sup_eggs_groundbeef / sup_eggs

# Print Confidence {eggs -> ground beef}
print(conf_eggs_to_groundbeef)
```

0.11127596439169138

Lift

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(Y)}$$

```
# Compute Lift {eggs -> ground beef}
lift_eggs_to_groundbeef = conf_eggs_to_groundbeef / sup_groundbeef

# Print Lift {eggs -> ground beef}
print(lift_eggs_to_groundbeef)
```

1.1325386823637411

4. Apriori Algorithm

Recap: Steps for Finding Frequent Itemsets

1	Prepare data and set minsup
2	Create a list of frequent itemsets (support ≥ minsup) of length 1
3	Create a list of itemsets of length 2 by combining the frequent itemsets of length 1
4	Prune itemsets whose support is less than minsup
5	Create a list of itemsets of length 3 from the pruned list
6	Prune itemsets whose support is less than minsup

- In the following, lengthen the itemsets and check whether "support ≥ minsup."
- Stop the process when you cannot create a list of frequent itemset.

Recap: Association Rule Selection

- Step 1. Generate rules from frequent itemsets
- Step 2. Select rules: Confidence ≥ minconf
- Step 3. Select rules: Lift > 1.0

Frequent Itemsets

```
# Import Apriori algorithm
from mlxtend.frequent_patterns import apriori

# Compute frequent itemsets
frequent_itemsets = apriori(onehot, min_support = 0.0005,
                            max_len = 4, use_colnames = True)

# Print number of itemsets
print(len(frequent_itemsets))
```

19788

```python
# Print frequent itemsets
print(frequent_itemsets.head())
```

```
     support              itemsets
0   0.020397             (almonds)
1   0.008932          (antioxydant
2   0.004666                juice)
3   0.033329           (asparagus)
4   0.004533             (avocado)
                      (babies food)
```

Computing Association Rule

```
# Import association rules
from mlxtend.frequent_patterns import association_rules

# Compute association rules
Rules = association_rules(frequent_itemsets,
                          metric = "support",
                          min_threshold = 0.005)

Rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(almonds)	(burgers)	0.020397	0.087188	0.005199	0.254902	2.923577
1	(burgers)	(almonds)	0.087188	0.020397	0.005199	0.059633	2.923577
2	(almonds)	(chcolate)	0.020397	0.163845	0.005999	0.294118	1.795099
3	(chcolate)	(almonds)	0.163845	0.020397	0.005999	0.036615	1.795099
4	(almonds)	(eggs)	0.020397	0.179709	0.006532	0.320261	1.782108
...	...	...	...	...	...	...	...
1935	(spaghetti, olive oil)	(pancakes)	0.022930	0.095054	0.005066	0.220930	2.324260
1936	(pancakes, olive oil)	(spaghetti)	0.010799	0.174110	0.005066	0.469136	2.694478
1937	(spaghetti)	(pancakes, olive oil)	0.174110	0.010799	0.005066	0.029096	2.694478
1938	(pancakes)	(spaghetti, olive oil)	0.095054	0.022930	0.005066	0.053296	2.324260
1939	(olive oil)	(spaghetti, pancakes)	0.065858	0.025197	0.005066	0.076923	3.052910

```
filtered_rules = Rules[(Rules['antecedent support'] > 0.01) &
                       (Rules['support'] > 0.009) &
                       (Rules['confidence'] > 0.5) &
                       (Rules['lift'] > 1.00)]

filtered_rules
```

	antecedents	consequents	antecedent support	consequent support
1406	(ground beef, eggs)	(mineral water)	0.019997	0.238368
1593	(ground beef, frozen vegetables)	(mineral water)	0.016931	0.238368
1737	(ground beef milk)	(mineral water)	0.021997	0.238368

support	confidence	lift
0.010132	0.506667	0.005365
0.009199	0.543307	0.005163
0.011065	0.503030	0.005822