Frequent Itemset and Association Rules Mining using Apriori Algorithm

In this part, you will build a system which can help make recommendations using the Apriori algorithm.

To solve this assignment you will need to go though these pages:

- https://rasbt.github.io/mlxtend/user_guide/preprocessing/TransactionEncoder/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/fpgrowth/

The `apply` function in `pandas` can prove very useful for this assignment. See https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html

Source: Online Retail. (2015). UCI Machine Learning Repository. https://doi.org/10.24432/C5BW33.

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

Load and Inspect Data

In [ ]:
```
invoices = pd.read_csv('apriori_data.csv')
invoices.head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0

Data Transformation

Drop everything except InvoiceNo and StockCode since we can use InvoiceNo for transaction id and StockCode for item name

In [ ]:
```
data = invoices[['InvoiceNo', 'StockCode']]
```

In [ ]:
```
data.head()
```

Out[ ]:

	InvoiceNo	StockCode
0	536365	85123A
1	536365	71053
2	536365	84406B
3	536365	84029G
4	536365	84029E

Group the data by InvoiceNo and create a list of StockCode for each invoice

In [ ]:
```
transactions = data.groupby(['InvoiceNo'])['StockCode'].apply(list).values.t
```

In [ ]:
```
transactions[0:4]
```

Out[ ]:
```
[['85123A', '71053', '84406B', '84029G', '84029E', '22752', '21730'],
 ['22633', '22632'],
 ['84879',
  '22745',
  '22748',
  '22749',
  '22310',
  '84969',
  '22623',
  '22622',
  '21754',
  '21755',
  '21777',
  '48187'],
 ['22960', '22913', '22912', '22914']]
```

Using TransactionEncoder, convert the transactions into a dataset where each row represents a transaction and each column represents an item. The values will be True or False depending on whether the item is present in that specific transaction.

In [ ]:
```
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
transactions_df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [ ]:
```
transactions_df.head()
```

Out[ ]:

	10002	10080	10120	10123C	10123G	10124A	10124G	10125	10133	10134	.
0	False	False	False	False	False	False	False	False	False	False	.
1	False	False	False	False	False	False	False	False	False	False	.
2	False	False	False	False	False	False	False	False	False	False	.
3	False	False	False	False	False	False	False	False	False	False	.
4	False	False	False	False	False	False	False	False	False	False	.

5 rows × 4070 columns

Use Apriori to get the frequent itemsets and inspect the results

Use apriori to find the frequent_itemsets for `min_sup` = `1%`

In [ ]:
```
frequent_itemsets =  apriori(transactions_df, min_support=0.01, use_colnames
```

In [ ]:
```
frequent_itemsets.shape
```

Out[ ]: (1087, 2)

In [ ]:
```
frequent_itemsets.head()
```

Out[ ]:

	support	itemsets
0	0.020193	(15036)
1	0.012587	(15056BL)
2	0.017876	(15056N)
3	0.011236	(16237)
4	0.012510	(20675)

Add an additional column called `items_count` to the dataframe which represents the number of items in the itemset.

In [ ]:
```
frequent_itemsets['items_count'] = frequent_itemsets['itemsets'].apply(lambd
```

In [ ]:
```
frequent_itemsets.head()
```

Out[ ]:

	support	itemsets	items_count
0	0.020193	(15036)	1
1	0.012587	(15056BL)	1
2	0.017876	(15056N)	1
3	0.011236	(16237)	1
4	0.012510	(20675)	1

Display the various itemsets generated sorted (descending) by the items_count.

In [ ]:
```
frequent_itemsets.sort_values(by='items_count', ascending=False).head()
```

Out[ ]:

	support	itemsets	items_count
1086	0.011699	(22423, 22699, 22697, 22698)	4
1085	0.010386	(21931, 22386, 85099B, 22411)	4
1084	0.010077	(20719, 22355, 20723, 20724)	4
1032	0.012548	(20725, 22384, 20728)	3
1024	0.011042	(20725, 22384, 20726)	3

Show how many itemsets exist by items_count

In [ ]:
```
frequent_itemsets.groupby('items_count')['itemsets'].count()
```

Out[ ]:
```
items_count
1    598
2    404
3     82
4      3
Name: itemsets, dtype: int64
```

Generate association rules

Generate all association rules using the `lift` metric with a minimum value of 2

In [ ]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=2)
```

In [ ]:
```
rules.shape
```

Out[ ]:
```
(1338, 10)
```

In [ ]:
```
rules.head()
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	
0	(20711)	(20712)	0.020541	0.033668	0.011158	0.543233	16.1350
1	(20712)	(20711)	0.033668	0.020541	0.011158	0.331422	16.1350
2	(21931)	(20711)	0.046371	0.020541	0.011506	0.248127	12.0798
3	(20711)	(21931)	0.020541	0.046371	0.011506	0.560150	12.0798
4	(20711)	(22386)	0.020541	0.047529	0.010888	0.530075	11.1526

In [ ]:
```
invoices.head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0

Add the names of the items back in the data frame as save all rules in a csv file

In [ ]:
```
rules['consequents_description'] = rules['consequents'].apply(lambda x: [inv
rules['antecedents_description'] = rules['antecedents'].apply(lambda x: [inv
```

In [ ]:
```
rules.head()
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	
0	(20711)	(20712)	0.020541	0.033668	0.011158	0.543233	16.1350
1	(20712)	(20711)	0.033668	0.020541	0.011158	0.331422	16.1350
2	(21931)	(20711)	0.046371	0.020541	0.011506	0.248127	12.0798
3	(20711)	(21931)	0.020541	0.046371	0.011506	0.560150	12.0798
4	(20711)	(22386)	0.020541	0.047529	0.010888	0.530075	11.1526

In [ ]:
```
rules.shape
```

Out[ ]: (1338, 12)

In [ ]:
```
# I used the following line to create the rules_100.csv file which only give
# rules.sample(100).to_csv('rules_100.csv', index=False)

# You must submit the rules.csv file that contains all the 1338 rules by run
rules.to_csv('rules.csv', index=False)
```

In [ ]:
```

```