

Lab 6: Association Rules

The objective of this notebook is to apply association rules with the **mlxtend** library ([official documentation](#)). It provides the implementation of the **Apriori** ([link](#)) and **FP-Growth** ([link](#)) algorithms. It also provides you a function to generate **association rules** ([link](#))

First, run the following cell to import some useful libraries to complete this Lab. If not already done, you must install them in your virtual environment

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder
```

```
In [2]: pd.set_option('display.max_columns', 75)
```

If the previous cell outputs one the following error: `ModuleNotFoundError: No module named 'mlxtend'`, then, you have to install the mlxtend package. If you don't remember how to install a Python package, please retrieve the guide on Anaconda-Navigator.

To install **mlxtend** you can use one of the following commands from the terminal of your virtual environment:

```
pip install mlxtend
conda install mlxtend
```

1. Apriori

Firstly, you will load the first dataset for this lab into a DataFrame `df`. The dataset is stored in the csv file from the following path `"data_lab6/Groceries data.csv"`. Then you will encode the dataset into a transactional dataset, apply the **Apriori** algorithm to extract the **frequent itemsets**, and generate the **association rules**.

Run the following cell to load the dataset.

```
In [3]: df = pd.read_csv("data_lab6/Groceries data.csv")
```

Now look the first 5 rows of the dataset.

```
In [4]: df.head()
```

```
Out [4]:
```

| | Member_number | Date | itemDescription | year | month | day | day_of_week |
|---|---------------|------------|------------------|------|-------|-----|-------------|
| 0 | 1808 | 2015-07-21 | tropical fruit | 2015 | 7 | 21 | 1 |
| 1 | 2552 | 2015-05-01 | whole milk | 2015 | 5 | 1 | 4 |
| 2 | 2300 | 2015-09-19 | pip fruit | 2015 | 9 | 19 | 5 |
| 3 | 1187 | 2015-12-12 | other vegetables | 2015 | 12 | 12 | 5 |
| 4 | 3037 | 2015-01-02 | whole milk | 2015 | 1 | 2 | 4 |

Run the next cell to remove duplicates in your dataset.

```
In [5]: df.drop_duplicates(inplace=True)# Dropping the duplicates
print(f"There are {len(df)} samples in the dataset.")
```

There are 38006 samples in the dataset.

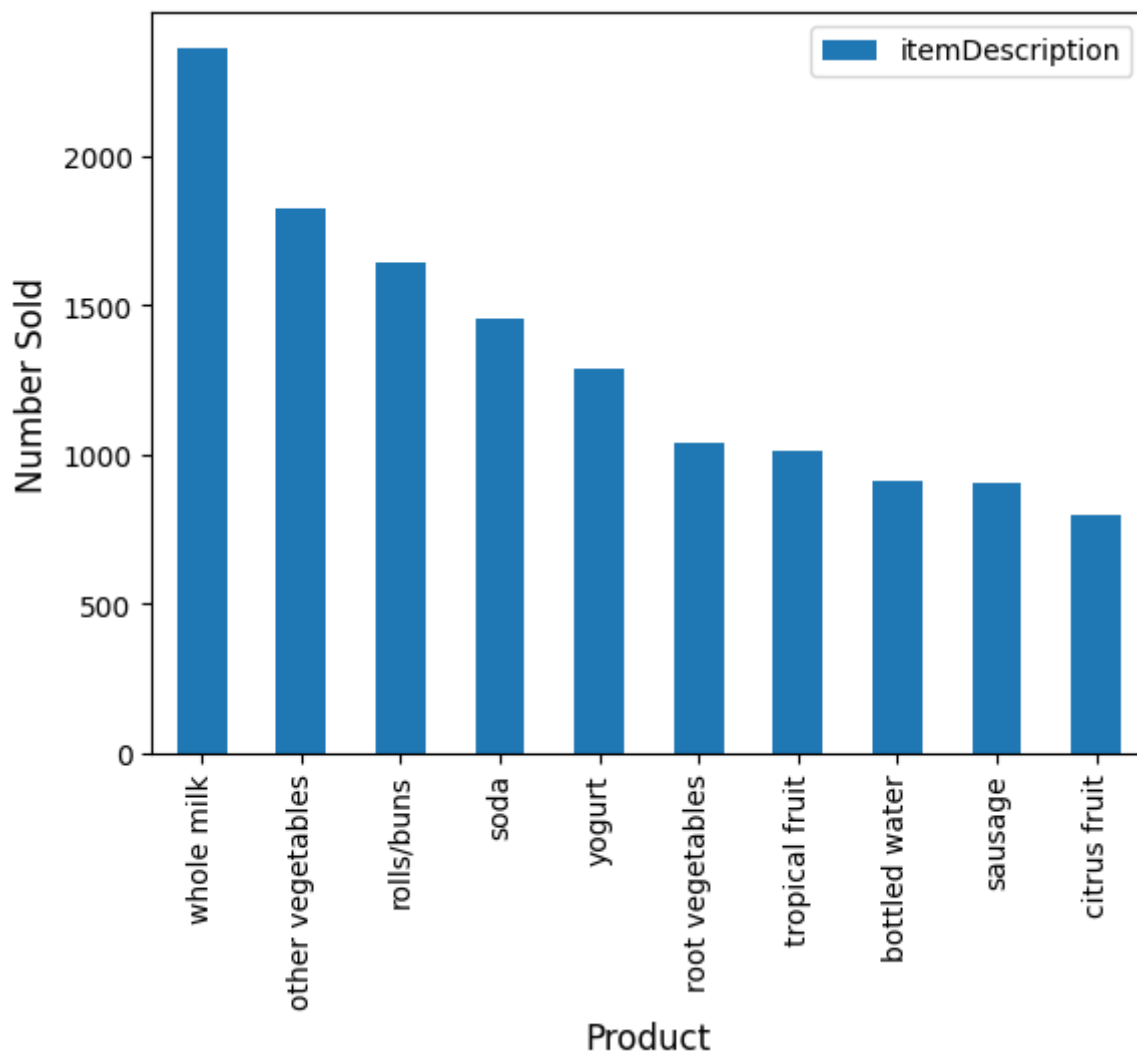
```
In [6]: df.columns
```

```
Out[6]: Index(['Member_number', 'Date', 'itemDescription', 'year', 'month', 'day',
            'day_of_week'],
            dtype='object')
```

Now, let's plot the most frequent 10 items.

```
In [7]: plt.figure(figsize=(5,3), dpi=150)
df.itemDescription.value_counts(ascending=False).reset_index().head(10).plot
plt.xlabel('Product', size= 12)
plt.ylabel('Number Sold', size= 12)
plt.show()
```

<Figure size 750x450 with 0 Axes>



Exercise 1.1

You will first create a transactional DataFrame. You want to analyze each product bought by a member on each date. However, the items bought are one in each row. Therefore, you should group all the items bought by a member on a given date with the `groupby` operation provided in Pandas. We didn't cover this kind of operation so far. The code is already provided to you.

The next cell creates a DataFrame of **transactions**. Specifically, it **groups all the items** for each `Member_number` and `Date` into a **list of items** in a new column `transaction`.

```
In [8]: df_transactions = df.groupby(['Member_number', 'Date'])['itemDescription'].a
df_transactions.head()
```

```
Out [8]:
```

| | Member_number | Date | transaction |
|---|---------------|------------|---|
| 0 | 1000 | 2014-06-24 | [whole milk, pastry, salty snack] |
| 1 | 1000 | 2015-03-15 | [sausage, whole milk, semi-finished bread, yog... |
| 2 | 1000 | 2015-05-27 | [soda, pickled vegetables] |
| 3 | 1000 | 2015-07-24 | [canned beer, misc. beverages] |
| 4 | 1000 | 2015-11-25 | [sausage, hygiene articles] |

Expected output

| | Member_number | Date | transaction |
|---|---------------|------------|---|
| 0 | 1000 | 2014-06-24 | [whole milk, pastry, salty snack] |
| 1 | 1000 | 2015-03-15 | [sausage, whole milk, semi-finished bread, yog... |
| 2 | 1000 | 2015-05-27 | [soda, pickled vegetables] |
| 3 | 1000 | 2015-07-24 | [canned beer, misc. beverages] |
| 4 | 1000 | 2015-11-25 | [sausage, hygiene articles] |

```
In [9]: print("Number of transactions:", len(df_transactions))
print("Number of distinct customers:", len(list(set(df_transactions["Member_
```

```
Number of transactions: 14963
Number of distinct customers: 3898
```

Exercise 1.2

Create a list containing all the transactions (i.e., list of list of items) into a variable `transactions`. Remember that all the transactions are stored in the `transaction` column of the DataFrame `df_transactions`.

```
In [10]: ##### START CODE HERE #####
##### Approximately 1 line #####
transactions = df_transactions["transaction"].tolist()
##### END CODE HERE #####

transactions[:5]
```

```
Out[10]: [['whole milk', 'pastry', 'salty snack'],
['sausage', 'whole milk', 'semi-finished bread', 'yogurt'],
['soda', 'pickled vegetables'],
['canned beer', 'misc. beverages'],
['sausage', 'hygiene articles']]
```

Expected output

```
[['whole milk', 'pastry', 'salty snack'],
['sausage', 'whole milk', 'semi-finished bread', 'yogurt'],
['soda', 'pickled vegetables'],
['canned beer', 'misc. beverages'],
['sausage', 'hygiene articles']]
```

Exercise 1.3

Encode the transactions into a **one-hot encoded transactional dataset**. Store the encoded dataset into a variable called `transaction_dataset`. You should first create a `TransactionEncoder()` object and then, call the `fit_transform()` method on your transactions (stored in the `transactions` variable). You can learn how to encode a list of transactions in a one-hot representation in the official documentation of the library ([link](#)).

```
In [11]: ##### START CODE HERE #####
##### Approximately 2 line #####
transaction_encoder = TransactionEncoder()
transaction_dataset = transaction_encoder.fit_transform(transactions)
##### END CODE HERE #####

transaction_dataset
```

```
Out[11]: array([[False, False, False, ..., True, False, False],
 [False, False, False, ..., True, True, False],
 [False, False, False, ..., False, False, False],
 ...,
 [False, False, False, ..., False, False, False],
 [False, False, False, ..., False, False, False],
 [False, False, False, ..., False, False, False]])
```

Expected output

```
array([[False, False, False, ..., True, False, False],
 [False, False, False, ..., True, True, False],
 [False, False, False, ..., False, False, False],
 ...,
 [False, False, False, ..., False, False, False],
 [False, False, False, ..., False, False, False],
 [False, False, False, ..., False, False, False]])
```

Exercise 1.4

Now, let's create a `DataFrame` from the one-hot encoded `transaction_dataset` and store the dataframe in a variable `df_transactions_onehot`. Set the columns of the dataframe to the `transaction_encoder.columns_` values.

```
In [12]: ##### START CODE HERE #####
##### Approximately 1 line #####
df_transactions_onehot = pd.DataFrame(transaction_dataset, columns=transacti

##### END CODE HERE #####

df_transactions_onehot.head()
```

Out [12]:

| | Instant food products | UHT-milk | abrasive cleaner | artif. sweetener | baby cosmetics | bags | baking powder | bathroom cleaner | beef | berries |
|---|-----------------------|----------|------------------|------------------|----------------|-------|---------------|------------------|-------|---------|
| 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 × 167 columns

Exercise 1.5

Apply the **Apriori** algorithm on the dataset stored in `df_transactions_onehot`. Set the value of the minimum support `min_support` to 0.01 and the value of the parameter `use_colnames` to `True`. Replace `None` with your code.

In [13]:

```
##### START CODE HERE #####
##### Approximately 1 line #####
df_freq_itemsets = apriori(df_transactions_onehot, min_support=0.01, use_colnames=True)
##### END CODE HERE #####

df_freq_itemsets
```

Out [13]:

| | support | itemsets |
|-----|----------|--------------------------------|
| 0 | 0.021386 | (UHT-milk) |
| 1 | 0.033950 | (beef) |
| 2 | 0.021787 | (berries) |
| 3 | 0.016574 | (beverages) |
| 4 | 0.045312 | (bottled beer) |
| ... | ... | ... |
| 64 | 0.010559 | (other vegetables, rolls/buns) |
| 65 | 0.014837 | (other vegetables, whole milk) |
| 66 | 0.013968 | (whole milk, rolls/buns) |
| 67 | 0.011629 | (soda, whole milk) |
| 68 | 0.011161 | (yogurt, whole milk) |

69 rows × 2 columns

Run the next cell to create a new column with the length of each transaction (i.e., the number of elements in each transaction).

In [14]:

```
df_freq_itemsets['length'] = df_freq_itemsets['itemsets'].apply(lambda x: len(x))
df_freq_itemsets
```

Out [14]:

| | support | itemsets | length |
|-----|----------|--------------------------------|--------|
| 0 | 0.021386 | (UHT-milk) | 1 |
| 1 | 0.033950 | (beef) | 1 |
| 2 | 0.021787 | (berries) | 1 |
| 3 | 0.016574 | (beverages) | 1 |
| 4 | 0.045312 | (bottled beer) | 1 |
| ... | ... | ... | ... |
| 64 | 0.010559 | (other vegetables, rolls/buns) | 2 |
| 65 | 0.014837 | (other vegetables, whole milk) | 2 |
| 66 | 0.013968 | (whole milk, rolls/buns) | 2 |
| 67 | 0.011629 | (soda, whole milk) | 2 |
| 68 | 0.011161 | (yogurt, whole milk) | 2 |

69 rows × 3 columns

Exercise 1.6

Print the frequent itemset with length equal to 2 (stored in `df_freq_itemsets`). You can use masking on the pandas array.

```
In [15]: ##### START CODE HERE #####
##### Approximately 1 line #####
df_freq_itemsets[(df_freq_itemsets['length'] == 2)]

##### END CODE HERE #####
```

Out [15]:

| | support | itemsets | length |
|----|----------|--------------------------------|--------|
| 64 | 0.010559 | (other vegetables, rolls/buns) | 2 |
| 65 | 0.014837 | (other vegetables, whole milk) | 2 |
| 66 | 0.013968 | (whole milk, rolls/buns) | 2 |
| 67 | 0.011629 | (soda, whole milk) | 2 |
| 68 | 0.011161 | (yogurt, whole milk) | 2 |

Exercise 1.7

Generate the association rules. Use `confidence` as metric and set the `min_threshold` to 0.01. Store the generated association rules in a DataFrame `df_ar`. Replace `None` with your code.

```
In [16]: ##### START CODE HERE #####
##### Approximately 1 line #####
df_ar = association_rules(df_freq_itemsets, metric = "confidence", min_thres

##### END CODE HERE #####

df_ar
```

Out [16]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | le |
|---|--------------------|--------------------|--------------------|--------------------|----------|------------|----------|------|
| 0 | (other vegetables) | (rolls/buns) | 0.122101 | 0.110005 | 0.010559 | 0.086481 | 0.786154 | -0.1 |
| 1 | (rolls/buns) | (other vegetables) | 0.110005 | 0.122101 | 0.010559 | 0.095990 | 0.786154 | -0.1 |
| 2 | (other vegetables) | (whole milk) | 0.122101 | 0.157923 | 0.014837 | 0.121511 | 0.769430 | -0.0 |
| 3 | (whole milk) | (other vegetables) | 0.157923 | 0.122101 | 0.014837 | 0.093948 | 0.769430 | -0.0 |
| 4 | (whole milk) | (rolls/buns) | 0.157923 | 0.110005 | 0.013968 | 0.088447 | 0.804028 | -0.0 |
| 5 | (rolls/buns) | (whole milk) | 0.110005 | 0.157923 | 0.013968 | 0.126974 | 0.804028 | -0.0 |
| 6 | (soda) | (whole milk) | 0.097106 | 0.157923 | 0.011629 | 0.119752 | 0.758296 | -0.0 |
| 7 | (whole milk) | (soda) | 0.157923 | 0.097106 | 0.011629 | 0.073635 | 0.758296 | -0.0 |
| 8 | (yogurt) | (whole milk) | 0.085879 | 0.157923 | 0.011161 | 0.129961 | 0.822940 | -0.0 |
| 9 | (whole milk) | (yogurt) | 0.157923 | 0.085879 | 0.011161 | 0.070673 | 0.822940 | -0.0 |

You can see that are presents the antecedents and consequents columns showing the antecedents and consequents of the rules, and several metrics as the support and the confidence.

2. FP-Growth

In this exercise, you will have to generate association rules for another dataset. This time you will have to use the **FP-Growth** algorithm. However, the necessary steps are the same as in the previous exercise. This time, the exercise has no hints. Only the code for loading the dataset is provided. However, it is important to start implementing from scratch without the structure, reading the documentation if necessary.

Please use the following parameter values: minimum support in frequent itemset generation 0.01 and minimum threshold in rule generation 0.05.

You can find the documentation with a very detailed example at the following [link](#).

```
In [17]: df = pd.read_csv("data_lab6/basket.csv")
```

```
In [18]: import csv
transactions = []

with open('data_lab6/basket.csv') as file_obj:
    reader_obj = csv.reader(file_obj)
    count = 0
    for row in reader_obj:
        if count != 0:
            trasaction = [x for x in row if x != '']
            transactions.append(trasaction)
        count += 1
```



```
In [19]: transactions[:2]
```

```
Out[19]: [['whole milk', 'pastry', 'salty snack'],
          ['sausage', 'whole milk', 'semi-finished bread', 'yogurt']]
```

Notice that the transactions are stored in a list of list.

```
In [20]: transaction_encoder = TransactionEncoder()
         transaction_dataset = transaction_encoder.fit(transactions).transform(transa
```

```
In [21]: df_transactions = pd.DataFrame(transaction_dataset, columns=transaction_enco
         df_transactions
```

```
Out[21]:
```

| | Instant food products | UHT- milk | abrasive cleaner | artif. sweetener | baby cosmetics | bags | baking powder | bathroom cleaner | beef | b |
|-------|-----------------------------|--------------|---------------------|---------------------|-------------------|-------|------------------|---------------------|-------|-----|
| 0 | False | False | False | False | False | False | False | False | False | |
| 1 | False | False | False | False | False | False | False | False | False | |
| 2 | False | False | False | False | False | False | False | False | False | |
| 3 | False | False | False | False | False | False | False | False | False | |
| 4 | False | False | False | False | False | False | False | False | False | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 14958 | False | False | False | False | False | False | False | False | False | |
| 14959 | False | False | False | False | False | False | False | False | False | |
| 14960 | False | False | False | False | False | False | False | False | False | |
| 14961 | False | False | False | False | False | False | False | False | False | |
| 14962 | False | False | False | False | False | False | False | False | False | |

14963 rows x 167 columns

```
In [22]: df_freq_itemsets['length'] = df_freq_itemsets['itemsets'].apply(lambda x: le
         df_freq_itemsets
```

Out [22]:

| | support | itemsets | length |
|-----|----------|--------------------------------|--------|
| 0 | 0.021386 | (UHT-milk) | 1 |
| 1 | 0.033950 | (beef) | 1 |
| 2 | 0.021787 | (berries) | 1 |
| 3 | 0.016574 | (beverages) | 1 |
| 4 | 0.045312 | (bottled beer) | 1 |
| ... | ... | ... | ... |
| 64 | 0.010559 | (other vegetables, rolls/buns) | 2 |
| 65 | 0.014837 | (other vegetables, whole milk) | 2 |
| 66 | 0.013968 | (whole milk, rolls/buns) | 2 |
| 67 | 0.011629 | (soda, whole milk) | 2 |
| 68 | 0.011161 | (yogurt, whole milk) | 2 |

69 rows × 3 columns

In [23]:

```
df_freq_itemsets = fpgrowth(df_transactions, min_support=0.01, use_colnames=
df_freq_itemsets
```

Out [23]:

| | support | itemsets |
|-----|----------|--------------------------------|
| 0 | 0.157923 | (whole milk) |
| 1 | 0.051728 | (pastry) |
| 2 | 0.018780 | (salty snack) |
| 3 | 0.085879 | (yogurt) |
| 4 | 0.060349 | (sausage) |
| ... | ... | ... |
| 64 | 0.011161 | (yogurt, whole milk) |
| 65 | 0.011629 | (soda, whole milk) |
| 66 | 0.013968 | (whole milk, rolls/buns) |
| 67 | 0.010559 | (other vegetables, rolls/buns) |
| 68 | 0.014837 | (other vegetables, whole milk) |

69 rows × 2 columns

In [24]:

```
#Let's view our interpretation values using the Associan rule function.
df_ar = association_rules(df_freq_itemsets, metric = "confidence", min_thres
df_ar
```

Out [24] :

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | le |
|----------|--------------------|--------------------|--------------------|--------------------|----------|------------|----------|-----|
| 0 | (yogurt) | (whole milk) | 0.085879 | 0.157923 | 0.011161 | 0.129961 | 0.822940 | -0. |
| 1 | (whole milk) | (yogurt) | 0.157923 | 0.085879 | 0.011161 | 0.070673 | 0.822940 | -0. |
| 2 | (soda) | (whole milk) | 0.097106 | 0.157923 | 0.011629 | 0.119752 | 0.758296 | -0. |
| 3 | (whole milk) | (soda) | 0.157923 | 0.097106 | 0.011629 | 0.073635 | 0.758296 | -0. |
| 4 | (whole milk) | (rolls/buns) | 0.157923 | 0.110005 | 0.013968 | 0.088447 | 0.804028 | -0. |
| 5 | (rolls/buns) | (whole milk) | 0.110005 | 0.157923 | 0.013968 | 0.126974 | 0.804028 | -0. |
| 6 | (other vegetables) | (rolls/buns) | 0.122101 | 0.110005 | 0.010559 | 0.086481 | 0.786154 | -0. |
| 7 | (rolls/buns) | (other vegetables) | 0.110005 | 0.122101 | 0.010559 | 0.095990 | 0.786154 | -0. |
| 8 | (other vegetables) | (whole milk) | 0.122101 | 0.157923 | 0.014837 | 0.121511 | 0.769430 | -0. |
| 9 | (whole milk) | (other vegetables) | 0.157923 | 0.122101 | 0.014837 | 0.093948 | 0.769430 | -0. |