Assignment 2

Mining association rules (Python version)

The objectives of this assignment are:

1) To understand the impact of the parameters on the number of frequent itemsets and rules extracted from the data.
2) To generate an interpretation of the rules and filter them, to identify the most relevant ones.

To this aim, we will use data from the AOL (America On Line) search engine, as described in the paper *A Picture of Search*, presented by G. Pass, A. Chowdhury, and C. Torgeson at the First International Conference on Scalable Information Systems, Hong Kong, June, 2006. In particular, for a subset of anonymous users in the original data we have listed the queries they submitted to the search engine for a period of three months, in 2006.

Preliminary of Language/Software: You can choose Pycharm or Jupiter Notebook (or others that you have been familiar with) as your Python IDE. To install packages, use the command `pip install [package name]` in the Windows' or Mac's terminal.

TASK 1: Warm Up) Read the www.csv data via Python and preprocess it (remember what you learned during Assignment 1 about preprocessing). This file can be found on Studium.

```
df = pd.read_csv('www.csv', sep='\t')
```

```
# check for outliers, missing values and duplicates (if any)

# for example, the following line of code returns any column of the dataframe that
has at least one null entry
df.loc[:, df.isnull().any()]
```

```
# remove duplicates and keep the first occurrence
print('Number of rows before removing duplicates:', len(df))
df = df.drop_duplicates('Query', keep='first')
df.reset_index(drop=True, inplace=True)
print('Number of rows after removing duplicates:', len(df))
```

Each row represents a search query. The first field is the anonymous user id, the second field contains the search query, and each of the following columns corresponds to a frequent keyword and takes the values **t**(rue) or **f**(alse) depending on the presence of that keyword in the search query. Count the number of records and the number of attributes (excluding User and Query).

TASK 2: Generation of Frequent Itemsets) As you know, you can split the Association Rule Mining task into two parts. First, you should find the frequent itemsets. HINT: You can choose to implement your own code or utilize relevant packages (like `pyfpgrowth`: https://pypi.org/project/pyfpgrowth/, and the function `find_frequent_patterns`). Note that you may need to transform the Query column accordingly, depending on the packages you choose (read the documentation).

For example, if you use `pyfpgrowth`, you can transform the Query column into a list of lists using the following code:

```
import pyfpgrowth
# Convert the Query column to a list of lists
associations = df['Query'].apply(lambda x: x.split()).tolist()
```

```
num_records = len(associations)
```

2.1) Set min-support so that an itemset must be present in at least 100 search queries to be considered frequent.
[Reminder: the *support* of an itemset is the fraction of records containing the items in the itemset (in this case, the keywords), and *min-support* is the minimum value of support to consider an itemset *frequent.* Hence, you can see the *min-support* as a threshold.]
2.2) Execute the process and examine the results.

```
sigma = 100
min_support = sigma / num_records
print(f"Minimum support: {min_support}")

# the function find_frequent_patterns takes SIGMA as second parameter
patterns = pyfpgrowth.find_frequent_patterns(associations, sigma)
patterns
```

```
# number of frequent itemsets found
num_frequent_itemsets = len(patterns)

# maximum size of frequent itemsets
max_itemset_size = max(len(itemset) for itemset in patterns)

print(f'Number of frequent itemsets: {num_frequent_itemsets}')
print(f'Maximum size of frequent itemsets: {max_itemset_size}')
```

```
#  the support of an itemset is the fraction of records containing the items in
the itemset (in this case, the keywords)
support = {key: value / len(df) for key, value in patterns.items()}
support
```

TASK 3: Effect of Support) Execute the previous process varying the min-support (by setting a for loop). Choose different values in the range [0.001, 0.01] and for each of these check:

3.1) the number of the found frequent itemsets, and
3.2) the maximum size of the found frequent itemsets.
3.3) From this analysis, choose a value for min-support leading to the discovery of around 150 frequent itemsets and use this value in the next tasks.

```
num_records = len(df)

# define range for min-support values (0.001 to 0.01)
min_support_values = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008,
0.009, 0.01]
```

```
for min_support in min_support_values:

    # the second parameter of the function 'find_frequent_patterns' is the support
count, NOT the min-support, so you have to calculate it
    support_count = min_support * num_records
    patterns = pyfpgrowth.find_frequent_patterns(associations, support_count)

    support = {key: value / len(df) for key, value in patterns.items()}
    print(f"Minimum support: {min_support}, Support count: {support_count:.0f}")
    print(f'Found {len(patterns)} patterns')
    print('Patterns', patterns)
    print('Support', support)
    print()
```

TASK 4: Generating the Rules) Taking the frequent itemsets as input, generate the association rules. For now, set a minimum confidence of 80%. HINT: You can choose to implement your own code or utilize relevant packages (like pyfpgrowth, and its function generate_association_rules)

4.1) Execute the process and inspect the result.
4.2) Find, among the identified rules, an example of a high-confidence rule X -> Y where Y -> X does not have high confidence. Why is the confidence of Y -> X lower?

```
min_support = # answer to task 3.3 to set a value for this variable
support_count = # remember the formula
confidence = 0.8

patterns = pyfpgrowth.find_frequent_patterns(associations, support_count)
rules = pyfpgrowth.generate_association_rules(patterns, confidence)
rules
```

TASK 5: Impact of Confidence) Execute again the previous process varying the min-confidence (by setting a for loop). Choose 5 different values of min-confidence in the range [0.1, 1] and for each of these indicate the number of generated rules. See how the number of rules changes.

```
min_confidence_range = [0.1, 0.3, 0.5, 0.7, 0.9]

for min_conf in min_confidence_range:
    rules = pyfpgrowth.generate_association_rules(patterns, min_conf)
    num_rules = len(rules)
    print(f"Minimum confidence: {min_conf}")
    print(f"Number of generated rules: {num_rules}")
    print()
```

TASK 6: Rule Interpretation) In this activity, you should test different evaluation functions with respect to their ability to highlight the *relevant* rules. Therefore, you are requested to provide your own subjective judgment on the identified rules and compare this against the values assigned by the algorithm:

6.1) Execute the process with a very low support, to extract a larger number of rules.
6.2) Choose up to 3 most **interesting** rules (notice that this is a subjective measure: you decide what is interesting). (Can you find 3?)
6.3) Choose up to 3 most **unexpected** rules (notice that this is also a subjective measure). (Can you find 3?)
6.4) For each of these 6 rules, provide their ranking according to confidence and lift (that is, at which position do the rules that you have selected appear, when rules are sorted by confidence/lift?). $\text{lift}(A \rightarrow B) = \text{confidence}(A \rightarrow B) / \text{support}(B)$
6.5) Describe (if possible) the features characterizing the relevant (interesting and/or unexpected) rules, and update your process so that irrelevant rules are no longer identified (still preserving the interesting ones).
HINT: If you implement your own code, the filter(s) can be embedded in the code of **TASK 4**; if you use package (like pyfpgrowth), you can add a filter after the code of **TASK 4**.