COMS 6111 - ADVANCED DATABASE SYSTEMS
Fall 2016
PROJECT 3

Project Group 26:
Gaurang Sadekar - gss2147
Tanaya Thaly - tpt2109

List of files submitted:
├─── DOHMH_New_York_City_Restaurant_Inspection_Results.csv (Original Data)
├─── INTEGRATED-DATASET.csv
├─── assoc_rules.py (Implementation of the Apriori Algorithm)
├─── clean_data.py (Generating database and INTEGRATED-DATASET.csv)
├─── db_utils.py (Utility functions for access to the sqlite3 database)
├─── baskets.sql (creation and deletion of the market baskets at every run)
├─── schema.sql (sqlite schema for mapping ids to actual values)
├─── test.db (sqlite3 db file)
├─── example_output.txt (Interesting run, Support = 1%, Confidence = 40%)
├─── output_s0.02_c0.5.txt (Support = 2%, Confidence = 50%)
├─── output_s0.005_c0.5.txt (Support = 0.5%, Confidence = 50%)

The Dataset:
We have used the **Department of Health and Mental Hygiene New York City Restaurant Inspection Results** data set. This is the data in the original dataset file `DOHMH_New_York_City_Restaurant_Inspection_Results.csv`. This holds data of restaurant inspections over time, through many different inspections.
There is a row in the dataset for each inspection of the restaurant and the violation that was discovered in the inspection. If a restaurant has multiple violations in one inspection, each of those violations creates a new row.
Out of all the columns, we are only using 3 columns, Borough, Cuisine and Violation Code (and description). There is a code for each description, so the columns essentially represent the same information. We keep the description only for displaying the violations properly rather than using the codes.

Why this dataset is interesting:
We believe this data set is interesting because it can provide insightful associations between all combinations of columns.

- We can find which cuisine is concentrated the highest in which borough, with rules of the type `[cuisine] => borough`. It can help to find what kinds of cuisine are best found in which borough.
- We can find which type of violation is associated with what type of cuisine with rules of the form `[violation] => cuisine`. For example, if a cold food storage violation is associated with restaurants serving sandwiches, this provides an important insight to health inspectors in future checkups to

inspect for this violation with more rigor at sandwich places rather than Indian restaurants (for example), allowing them use their time judiciously.

- Another interesting pattern that can be observed is `[violation] => borough` to determine if violations are localized to boroughs. This can help to identify if violations are associated with external problems within boroughs. For example, New York City is notorious for its vermin problems. Hypothetically, If we find that a facility being vulnerable to vermin is associated with Brooklyn (with high confidence) but not with Manhattan, it localizes the problem to Brooklyn and helps the DOHMH address it at a more grassroot level.
- Last but not least, finding rules of the form `[violation, cuisine] => borough` helps the DOHMH to isolate violations with restaurants in a borough. This helps the DOHMH to manage health violations in a more granular way, and allocate resources to inspect and address these violations in a localized manner by borough. For example, if we find Plumbing problems in American restaurants in Queens, and a violation with flies in American restaurants in Manhattan, this helps DOHMH provide health recommendations to American restaurants in different boroughs differently.

Data Processing:

The total number of transactions in the data are 437022.
Since each of the columns are strings, we use 3 tables, `boroughs (id, name)`, `cuisines (id, name)` and `violations (id, code, description)`, to associate these strings with unique IDs. The schema for these tables is in the file `schema.sql`.

There are 6 possible values for boroughs (Manhattan, Brooklyn, Queens, The Bronx and Staten Island, and another value for 'Not specified')
There are 84 values for cuisine types in the data set (including one for N/A)
There are 97 types of violations in the data set (including No Violation and N/A)

To generate the INTEGRATED-DATASET file, we add each unique borough, cuisine and violation code into the database, and get the integer ID associated with them. We discard all the other columns of the data. To create a lexicographic ordering on the data, borough IDs are from 1 - 6, cuisine IDs are from 101 - 184 and violation IDs are from 201 - 297.
Thus, every row in INTEGRATED-DATASET.csv has 3 columns, where the first column denotes borough, the second denotes cuisine and the third denotes the violation.
The code to transform the data and create INTEGRATED-DATASET.csv is in `clean_data.py` using many functions from `db_utils.py`. The data is persisted in `test.db`.

How to run the program:

To get the association rules from INTEGRATED-DATASET.csv, simply run the following command:

```
> python assoc_rules.py <path to INTEGRATED-DATASET.csv> <min_sup> <min_conf>
```
The min_sup and min_conf values are expected in decimal. For eg., if you want to use 1% support and 50% confidence, supply the values 0.01 and 0.5.
This creates the `baskets` table in `test.db` and uses the persistently stored boroughs, cuisines and violations for outputting the results. `test.db` is included in the submission with these tables. However, the transactions are read from the dataset file and the `baskets` table is newly created and populated in each run of the code independently. The SQL code to create the `baskets` table is in the file `baskets.sql`.

Internal Design of the System:
We have implemented the Apriori Algorithm in 2.1 of the paper Fast Algorithms for Mining Association Rules by Agrawal et al. to find the large itemsets and high confidence association rules.

The program takes 3 command line inputs:
- INTEGRATED-DATASET.csv
- Minimum Support Threshold
- Minimum Confidence Threshold

There are 2 parts to the program:
1. Find large itemsets
2. Find high confidence association rules

Finding Large Itemsets:
We start with frequent itemsets of size 1, which we find by going through each id in each of the 3 tables, boroughs, cuisines and violations and check if they meet the minimum support threshold. This way, we get a list of frequent itemsets of size 1.

Since we have 3 columns, can build large itemsets of size at most 3, where each element is of a different category. From the large itemsets obtained from the previous round, we generate candidates using the `candidate_gen` function. To generate candidates, we use the steps mentioned in section 2.1.1 of the Agrawal paper. The steps are as follows:
1. **Merge:** We take all 2 combinations of the previous itemsets and perform the following steps:
    a. We first flatten each combination by taking their union. If we are generating itemsets of size 3, we will take all 2 combinations of itemsets of size 2 and take their union. This leaves us with candidates of size 3 or 4 (since we take combinations without replacements).
    b. We first filter the combinations based on size, to keep only candidates of the required size.
    c. The next step is to remove duplicate sets from the candidates.
    d. The last check is to to remove sets that have duplicate columns. No transaction has 2 values of the same category, thus these are not valid candidates for itemsets. This is variation of the join step mentioned in

the paper, where we keep candidates that have all but one column different.

2. **Prune:** Here, we take the k - candidates remaining after the merge step and remove every candidate *c* such that there exists a (k - 1) subset of *c* that is not a frequent itemset of size (k - 1). The sets that remain after pruning are the candidates of size k.

For each candidate of size k, we filter those whose support is greater than the minimum support threshold. The support of an itemset is obtained by running a count(*) query on the baskets table where the relevant ID is equal to the value in the itemset and dividing by the total number of transactions in the baskets.
The itemsets that remain are the frequent itemsets of size k. These are added to a list of large itemsets. This gives us the large itemsets present in the data.

Finding High Confidence Association Rules:

To find the association rules, we first sort each frequent itemset in reverse lexicographical order, so that we can get the rules from [violation] => [cuisine], [violation] => [borough], and [cuisine] => [borough] etc. Since there are only 5 boroughs but 84 cuisines and 97 violations, we observed that it is impossible to find association rules that go from [borough] => [cuisine] etc because each borough has a very high support as compared to any 2 - itemset containing a borough. No rule containing [borough] on the LHS met a reasonable confidence threshold (>= 40%).
Since we are only interested in itemsets that have one item on the RHS, for each itemset of size > 1, we take all items except the last item as the LHS. The confidence of the rule [LHS] => [RHS] is given by:
conf([LHS] => [RHS]) = support(LHS U RHS) / support(LHS)
If this confidence >= the minimum confidence, we save this rule as a high association rule and record its confidence.

Interesting Observations from Itemsets and Rules:

Our data is very varied, so expecting a very high support seems unlikely. We first started with a min. support of 5%. We observed that there no 3-itemsets that were frequent, and we only get those itemsets which we expect to be very frequent such as all the boroughs (Manhattan has the most restaurants and inspections by a large margin) and American and Chinese cuisines.
Violations such as ‘facility not vermin proof’ and ‘evidence of mice present’ are found in the frequent itemsets with supports 10% and 7% respectively. This is a good baseline to start, as these itemsets are expected. It also confirms that NYC really does have a vermin problem, as almost 1 in 10 restaurants either has mice or is vulnerable to them.

Relaxing the support to 2% and confidence to 50% gives us some more itemsets to work with, but still does not give frequent 3-itemsets. The association rules we find are not very surprising with rules such as: Japanese restaurants, Italian restaurants and Cafes are present and inspected in Manhattan over 50% of the time. This also aligns with common knowledge.

Tuning the support to 1% and confidence to 40% shows interesting results. This is included in the `example_output.txt`. To recreate those results, run the following:
`> python assoc_rules.py <path to INTEGRATED-DATASET.csv> 0.01 0.5`

We find that if a restaurant serves French food, with a 78% confidence it is located in Manhattan. This rule is much higher in confidence than any other *cuisine => borough* rule obtained. We find other interesting `[violation, cuisine] => [borough]` rules such as:
[Improperly constructed non - food surfaces, surface not spaced or movable, American cuisine] => Manhattan - This tells us that small restaurants (not enough space to keep clean) serving American cuisine are more likely to be in Manhattan.
[Facility not vermin proof, American cuisine] => Manhattan - If an American restaurant violates vermin proofing, it is located in Manhattan over 50% of the time.

These are interesting insights to both the DOHMH and restaurant owners. It tells the DOHMH to check American restaurants in Manhattan for rats and general cleanliness first. This configuration gave us the most prominent rules present in the dataset. To find some more detailed rules, we ran the algorithm with one final configuration.

Support 0.5%, Confidence 50%:
Using a lower support, we hope to find more 3-itemsets and generate more rules. This is still a valid candidate for the support threshold, because the total dataset is about 460,000 rows, so even if we identify a violation and cuisine patten localized in 2500 of those inspections can help the DOHMH allocate resources for inspections and cleanups in a much more effective way.
Here, *[Korean] => Queens* is unexpected because New York's Koreatown is located in Manhattan, where a large number of Korean restaurants are concentrated.
We also observe that most `[violation] => borough` rules involve Manhattan, and all `[violation, cuisine] => borough` rules involve American restaurants in Manhattan. This implies that although Manhattan might be checked more frequently than the other boroughs, but its violations are still the most prevalent.

The conclusive insight we get from these rules is that although there are more restaurants in Manhattan, inspection shows that they violate the rules more frequently. It is useful to the DOHMH to find that it is beneficial to impose higher standards of compliance with restaurants in Manhattan. For those interested in experiencing different cuisines, the `[cuisine] => borough` rules tend to indicate that for the most widely consumed cuisines (American, Chinese etc), Manhattan is the best bet, but for the more niche cuisines (Jewish, Caribbean), other boroughs have more variety.