########## Task: mark duplicate entries in datasheet ##########

########## Solution Design ##########

This task is about dealing with a datasheet. I have chosen Python as the tool since Python provides powerful libraries for string processing.

By observing the datasheet I have noticed that the zipcode and the address are used to locate the companies. Since there are limited number of zipcode-address variations and they are harshable, it is easy to utilize the dictionary of Python to nicely structure the data. One important thing is that the zipcode-address combinations are exact strings, rather than fuzzy names like the company names. This makes sure that the zipcode-address combinations are comparable through the internal hash function of the Python dictionary.

My solution takes the original data and uses the zipcode-address combinations as the dictionary keys. The value corresponds to the dictionary key is set to be a list, with each item of the list holding one entry of the company information. To keep track of the original order of these entries, an index is attached in the begining of the entry string.

To search for the duplicates, a nested control loop is used. simply put, if a key of the dictionary holds multiple entries(multiple companies with the same address), the company names of these entries are compared. if two companies' names have a similarity of over 75 percent, one company entry is marked.

The mark convetion is as follow:

if the two duplicates are of no exception(one CustomerNo starts with Abo, another with GS), one entry is attached with a new string as "DUP\_linenumber", where linenumber is the line number of the other duplicate entry in the origianl datasheet. If the customerNos are of exception, then one duplicate is marked by "DUPYellow\_linenumber". The new marking string is attached at the end of an entry

. after the duplicates matching, a newfile is generated, with duplicated company entries marked.

For instance, the following entry is the duplicate of entry line number 2110:

AA24FDB9-EC50-4C43-B2E9-F6B02AA87451 Blum Recycling GmbH NULL DE 18211 Admannshäger Damm 18 Bargeshagen NULL NULL NULL NULL Abo24023 DUPYellow\_2110

Entry of line number 2110 is shown as follow:

D3B44513-EB04-482E-B90B-4F2D644A3EE5 Blum Recycling GmbH NULL NULL DE 18211 Admannshäger Damm 18 Admannshagen-Bargeshagen NULL NULL 038203 704-0 038203 704-40 Blumrec@blume-gruppe.de http://www.blum-gruppe.de GS310210

For searching duplicates, python's standard library 'difflib' is imported. the function used for calculating string similarity is SequenceMatcher(). There is a third party libaray, which may perform more accurately and reliably. Since it requires installation on a local machine, I did not use this third party lib.

For companies with a name similarity of between 50% to 75%, the duplicates pairs are saved in a list and then write to a file. Within this file, companies with similar names are grouped together. Eache line of the file consists of several numbers, which are index numbers for addressing the entries in the original datasheet. This file is named as index.txt.

All previously mentioned functions are implemented in processor3.py, which is my 3rd version of the datasheet processor.

Since it can take a long time to manually match the remaining entries, another script is written. By taking in the processed datasheet and index.txt, the script ask the user to match the similar company names. If the user decides to resume the work later, since the work can take time, he can press ctl+C to send SIGINT. this signal will be captured and the program will save the match work that's been done and update the processed datasheet and the index.txt. After each round of manual matching, matched entrie's line numbers will be removed from index.txt, and the duplicated entries in the processed datasheet will be marked.

Different than the machine matching, human matching entries will be marked as "HDUP\_linenumber" and "HDUPYellow\_linenumber".

########## Instructions: ##########

put scripts under the same directory of the datasheet.

under shell, use processor3.py to generate the marked file and the index file.

Then use humanprocessor.py to manually mark the remaining undetermined duplicates.

Net result is saved in the datasheet file with the name prefix of Processed

########## Usage example: ##########

Under shell, type in "python processor3.py Sub-Companies.csv" to perform machine matching

After machine matching, do not remove the generated files. Then call the humanprocessor script to do manual matching.

for example. "python humanprocessor.py Processed\_Sub-Companies.csv index.txt"

Use ctrl+C to end current work. things will be backuped properly.

To resume unfinished matching work, simple do the same "python humanprocessor.py Processed\_Sub-Companies.csv index.txt", the matching will go on.

Notice! matching work is saved based on address group, meaning if the manual matching of one group of entries are not finished, they are not saved. However, you can still resume the work since the unfinished group will not be removed from the index.txt.

More details are commented in the source file of humanprocessor.py

########## Test done ##########

processor1.py is the first version of the solution. Since it does not use any particular data structure and performs a simple nested loop, the performance is really bad. it took about 2 minutes to finish the machine match up.

Processor3.py and humanprocessor.py works together, to present my final solution. performance is much better and functionalities are more complete.

for more complex datasheet, a data base might be considered. However, making use of the simple hash table based dictionary performs well enough in this task.

########## Appendix ##########

########## Third party library (not in use this time) ##########

http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ #

https://github.com/seatgeek/fuzzywuzzy #

git clone git://github.com/seatgeek/fuzzywuzzy.git fuzzywuzzy

cd fuzzywuzzy

python setup.py install

Usage

>>> from fuzzywuzzy import fuzz

>>> from fuzzywuzzy import process

Functions:

Simple Ratio()

Partial Ratio()

Token Sort Ratio()

Token Set Ratio()