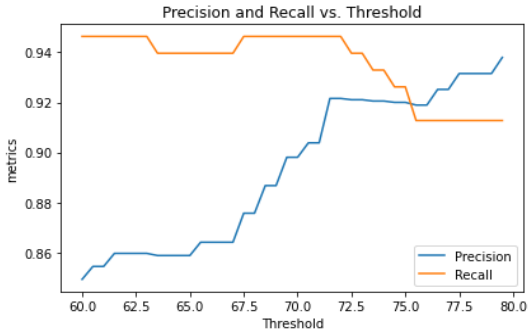
Quang Tri Le – 1118563

Report for Task 1

Task 1A:

Method:
The algorithm includes a double loop that will execute all possible comparisons. In order to determine whether two products are the same, product's **description** in *abt_small.csv* was compared with product's **name** in *buy_small.csv*. Comparison between product's **name** in *abt_small.csv* and product's **name** in *buy_small.csv* was also experimented. However, the result for this comparison was not as ideal as when using product's **description** in *abt_small.csv*. This could be explained by the fact that some product details appear in both abt's description and buy's name but not in abt's name. Before these two strings were brought to comparison, they were preprocessed by removing all punctuations, stop words and normalized to lower case.

After being cleansed, two products' **brand** are extracted for early comparison. If two products have different brands, the comparison halts for this pair and continue to next pair (this process helps algorithm end earlier). If the two products have the same brand, their product codes are extracted for the next filter. The product code in the **name** of *abt_small.csv* product is extracted by getting the last word of the string. This code is then trimmed by one last letter because product codes from two files could be different at some final letters. If this trimmed code exits in the name of *buy_small.csv* product name, the two products are surely the same (there is a very small number of exceptions).



If product code does not exist in product name of *buy_small.csv*, two product's strings were brought to the final scoring function which is direct string comparison.***)***. For products name comparison, comparing two sets of words would be more useful than exact string match comparison since two products from different shops could have different ways to express product's features. Therefore for this process, ***fuzzywuzzy*** module was used along with method ***token_set_ratio(abt_description, buy_name)***. The threshold for this method which determines whether two products are the same or not was tested in a wide range (60-80). Threshold with value of **72** was chosen to ensure both recall and precision are above 0.9.

Evaluation:
The result for applying above filters and comparisons are realistically reliable with **Recall of 0.946** and **Precision of 0.922**. Without early preprocessing and comparisons, if the algorithm relies solely on direct string comparison (*token_set_ratio*), the metric for recall would be extremely low due to blindly set comparison. In early stage, product code comparison guarantees substantial number of true positive that could not be identified by using final scoring function along with its threshold. Besides, if brand comparison were not taken place, there would be some extra false positive which would lower the precision.

Recommendation:
Although the performance metrics are reasonably high, there is still room for improvement. The product code extraction from the name of abt's product is not general enough since it only considers the last word of the string. Therefore, the algorithm could fail if product code stores in the middle of the string. Such problem can be solved by using **regex** which extracts the word that is made up of capital letters and numbers. On the other hand, text preprocessing could also be improved by considering part of speech taggers for cleaner lemmatization and recognizing abbreviation such as "ft" (feature)…

Task 1B:

Method:
For blocking implementation, product **name** from both *abt.csv* and *buy.csv* are used to extract block keys. Before block keys were extracted, the product name had to be preprocessed with the same method in *task 1a*. However, words in product name are more meticulously lemmatized by considering **part of speech tagging** in order to create consistency in block keys.

After cleansing, block keys for each product are extracted by taking the combination of two words (using **combinations** module) from the cleaned product name. This is a form of bigram without considering the order of words in a sentence. By not considering the order of the words, block keys can cover the majority of unstructured product name. After this stage, each file will have their own set of block keys associated with the correct product ids (a dictionary with block key as the key and an array of product ids as value).

The unified set of block keys is made by performing inner merge between two independent sets of block keys. With this unified set of block keys, the product ids from each file will have a reference for which block keys they belong to.

Evaluation:
Implementation of the above method yields **Pair Completeness of 0.975** and **Reduction Ratio of 0.942**. The metrics are reasonable since product names were more carefully preprocessed (with assistance of part of speech taggers) and "combination of two" method can generalize the name of product from both files.

Time complexity of the algorithm is O(m+n) with m and n being number of records in *abt.csv* and *buy.csv* respectively:

- Independent blocking for *abt.csv* and *buy.csv* is roughly O(m+n) assuming text-preprocessing and adding block keys were done in constant time.
- Assuming inner merge of two set consumes constant time: O(1)
- Unified blocking with same set of block keys is O(x) with x being number of mutual block keys
- No comparison was conducted

Thus, time taken to produce this blocking implementation is linear in the number of items in *abt.csv* and *buy.csv*.

However, inner merge cannot absolutely guarantee two actual same products from two files having exact same blocks due to noise (such as numeric value keys) that were taken from some random product name which only exists in either product of the pair.

Recommendation:
Potential improvement for this algorithm is considering trigram (without assessing the order) in order to better generalizing the block keys. Besides, numeric values (such as: "120W", "1080" …) could be removed in order to reduce noise and increase blocking qualities.