Task 1C

i. To process the two datasets, the csv's were first imported into *Pandas* dataframes. The common and relevant columns are then extracted from both datasets, namely the ID, name, description and price of the products. Before data linkage can be done, pre-processing was performed on the price of the products to remove dollar signs using regular expressions. Then, each item from '*abt_small.csv*' was used to loop with each item from '*buy_small.csv*', to compare two items. Common punctuations were first removed from their names, and they were converted into lower case and tokenized using *word_tokenize* from *nltk*. In choosing the similarity functions between names, token-based algorithms are more suitable for string of words than edit-based. Hence, *cosine_normalized_similarity* was used on two strings (from the *textdistance* library). The similarity between the names is then computed. We also extract the brand and model from the tokenized product name, as product name is observed to appear mostly in the first position and model name in the last (if any). We then compare the brand and model and assign similarity scores respectively. The absolute price difference is also calculated and assigned scores. The scoring function ***f*** is *0.5*(name score) + 0.2*(model score) + 0.2*(brand score) + 0.1*(price score)*, ranges between [0,1]. Model scores and brand scores have default values of 0.2 and 0.0 and are assigned 1 respectively if brand matches or model (assumed to be alphanumerical characters) in one product can be found in another. Price score is 0.5 if *\$10 < price difference < \$50*, 1.0 if below range and 0.0 if above range. Similarity scores between one product from 'buy' and all products from 'abt' were ranked and the pair with the highest score was added into the match list if it exceeds the threshold θ_1. A list was also built to filter out products of 'abt' that have previously in a highly similar match (similarity > θ_2) to reduce duplication. If the top 'abt' product in the loop has previously been in a highly similar match, the second-highest match is added into the match list provided the similarity > θ_2. After experimenting with different thresholds, θ_1 = 0.55, and θ_2 = 0.7 was found to produce the best F1 score with the most reasonable number of linked records. Then, the ID pairs in the match list were written into '*task1a.csv*'.

ii. This product comparison algorithm yields a *recall* of 0.812 and *precision* of 0.840. The number of true negatives was highest, meaning that the algorithm managed to differentiate between completely different products. However, there were some false positives, meaning that the similarity function failed to pick up some of the more nuanced differences in product names (e.g. same product name different models: "Panasonic Corded Phone - KXTS3282B", "Panasonic KX-TS208W Corded Phone"). There were also some false negatives, which indicates that the algorithm wasn't able to recognise the same products with different product name details (e.g. same model but different model format: "GE Futura Indoor TV Antenna - Silver Finish - TV24746", "Ge 24746 Futura(tm) Indoor Hdtv Antenna"). Overall, the algorithm performed well on picking up most matches, but failed to pick up some details. The false negatives and positives were much lower compared to true positive and negatives. To improve the performance, we could first examine attributes such as description, and calculate the similarity between description strings, assigning a default value if not exists, and account into our scoring function. We could also try and average the scores with other similarity measures that work well with strings of words such as Jaccard Similarity, to pick up more similarity/differences between strings. Also, we could use the *recordlinkage* library to assist with linking data.

iii. The two datasets '*abt.csv*' and. '*buy.csv*' were opened and loaded into dataframes, then important features were extracted and cleaned using regular expressions. To place each item into blocks, a for loop is used. It first gets the product name, tokenises it using the *nltk* library, and removes punctuations and stop words from the word list, to avoid large, meaningless blocks. Then, each word in the word list is lemmatised using *WordNetLemmatizer* from the *nltk* library, and a block is created for each word in the processed product name. Before adding into the final list, some filtering works were also done to filter out potentially large blocks with generic product names or properties. Upon examining both datasets, colours and popular brand names were found to be creating unnecessary large blocks that will make the data linkage unproductive. Hence a small list including "sony", "black" was created to help with filtering out the blocks. Finally, we add the block key (which is the word) and the product ID into the csv file. The same process was done with both '*abt.csv*' and. '*buy.csv*'. At the end of the code, each product was assigned to at least one block, with the block key being a component of its name.

iv. This blocking method produced a *pair completeness* of 0.965 and *reduction ratio* of 0.931. The time complexity in this method is O(m+n), with m, n being the size of each dataset. This means that all record is assigned to blocks in linear time, and the algorithm only goes through each product once. Most matching records were put in the same block, producing high true positives (1059) and low false negatives (38), hence pair completeness is high. This was done by creating a large number of blocks, and by this most blocks having a low number of products assigned to it were able to be compared quickly and found matches. Most products that do not match with each other were able to be in different blocks, producing high true negatives. There is still a significant number of false positives, of some product pairs with common generic properties in their name, these were placed into blocks with a rather high number of elements, such as 'series'. However, the false positives (80392) are still much less than true negatives (1098963), hence when high true negatives added with low false negatives, the reduction ratio calculated was still rather high (*reduction ratio = 1 − (tp + fp)/n = (tn + fn)/n, n=fp+fn+tp+tn*). To improve the performance, we can consider creating blocks with multiple n-words, price, or a combination of both. This should reduce the number of pairs in the blocks, thus speeding up the comparison. We could also examine and add the description to the blocks in addition to name and price, if exists, to improve accuracy.