Overall, I matched products using their product code. For example, 'AM53BK' is the product code for the first product in 'abt'. I stored two kinds of fuzzy matching score in order there is no exact match of the product code. For each product in 'abt', I extracted its code by finding the last word in the column 'name'. Also, name of each product is stored as 'abt_info' and removed the punctuation for fuzzy match. Before the exact match, I stored some scores for fuzzy match. For each product in 'buy', name and description (if it is not none) are stored as 'buy_info' and removed punctuation. For the matching function, 'fuzz.token_set_ratio' is used for compare 'abt_info' and 'buy_info' and the score greater than threshold 'matchscore' is stored into a dictionary with key being 'idBuy'. Threshold is selected by testing program's precision and recall. To achieve better performance, I have calculated F1 score for each number I have chosen and kept the threshold with maximum F1 score. Then I find potential matching code from 'buy_info' because some of the matching pair have different but similar code. For example, 'ABS60BK' in 'abt' has code 'ABS60' in buy. I tokenized 'buy_info' and used 'fuzz.token_set_ratio' to store potential code greater than threshold 'matchscore_code'. At last, exact matching is realized by 're.search'. If there is no exact match, then I matched the potential code with the highest score. If there is no potential code, the most similar product information is matched.

The comparison method splits comparison into three parts which are exact match, fuzzy code match and fuzzy information match. Exact match is designed to improve the precision since if two product codes are the same, possibility which they are the same product is big. Fuzzy code match and fuzzy information match is designed to improve recall of the algorithm. Since there are matching products with different code or matching products without code information. There exist chances to improve that 'token_set_ratio' is not sufficient for evaluating the similarity of the products information, could combine other measuring method and calculate weighted score. Also, false positive occurs when two different products of the same brand have similar code which my algorithm could identify them as a match. For example, 13954 in 'abt' and 90105056 in 'buy'. Maybe the algorithm could specify different situations: same brand similar code, same brand similar code and very similar product information and same brand no code but very similar information. In this way, more matching pairs could be found and recall could be improved.

Overall, I used product brands as blocks to implement my blocking method. A set of blocks contains brands is created first. I extract the first word from the 'name' column from 'abt' which the product name is written in a formal way starting with brand name. At the same time, some pre-processing of the data is applied which I made all the brand into lower case. Because this could prevent the situations that pair not in the same block because of the formatting. Allocation of blocks for products in 'abt' is realized by using the lower case of the first word of each product's name. While for products in 'buy', I first check if the manufacturer information is missing, if is not, I matched the first word of 'manufacturer' with the brand set because this word stands for brand name in most cases. Function 'process.extractone' is applied to match this word with brand set. Otherwise, brand name should appear in products' name. To extract brand from it, I compare each word of product name with brand set, if found the word in brand set, then this word is the block.

I have tried several different methods of allocating each product to a brand block. At first, I thought it could be good enough to allocate each product directly by just extracting its brand from 'manufacturer' or 'name'. However, some of the products have missing data which may decrease pair completeness. Therefore, a set of blocks which contains relatively complete brands' names need to be created first. This method improves pair completeness since it handles products have missing information or informal brand names which make the blocking method more accurate by improving the number of true positive pairs. Also, when allocating a block for products in 'buy', when 'manufacturer' information is missing, I compare the whole product name with brand set which could also improve true positive pairs since some of the brand is inside the name rather appear at the beginning of it. However, it also decreases reduction-ratio to some degree since it adds more comparison. Meanwhile, the method could be improved. I have assumed that for 'manufacturer' column in buy, the first word is always its brand name. However, special cases exist where the first word is not brand name. For example, '202473822' in 'buy.csv' has brand 'Sony' which appears in 'name' column. Maybe the method could be improved by matching each product information of manufacturer and name together with the existing brand set.