The code for this solution is at : https://github.com/suraajsamanta/MachineLearningNLP-Model/tree/main

Solution Outline:
This solution has a few steps:

1. Cosine Similarity
   Cosine similarity takes the angle between two non-zero vectors and calculates the cosine of that angle, and this value is known as the similarity between the two vectors. This similarity score ranges from 0 to 1, with 0 being the lowest (the least similar) and 1 being the highest (the most similar). It uses this formula:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

2. Vectorizing our Words
   The first task is to define a function that can take words as inputs and convert them into a vector representation. This is done in the word2vec function.
   First, we use the Counter() method from the collections library to count the number of occurrences of each letter in our inputted word, as shown in the count_characters variable. This is just a dictionary where each key represents a character, and its value is the number of occurrences of that character.
   We then get the set of characters used in our inputted word by using the set() function so that we can get the intersection of characters between two words, and we'll name that variable set_characters . This is needed for later when we're calculating the cosine similarity.
   The final part is to calculate the 'length' of the word, which is the magnitude of the word vector as shown previously in the cosine similarity formula. Remember that the reason we calculate the length this way is because we're representing these words as vectors, where each used character represents a dimension of the vector.
3. Cosine Similarity Function
   Next, we have the cosine similarity of all of our pairings by comparing the similarity between the vector translated versions of our strings.
   First, we need to get a list of characters that are common to both words by using the intersection() function, and we call this variable common_characters .

Referring back to the formula, we then have to calculate the dot product. This is done by iterating through the list of common_characters and getting the count value of each iterated character, as seen in the product_summation variable.
Then, we calculate the product of the lengths of both vectors and call this variable length (looking back, I probably should have used a less confusing variable name).

4. Finding Similar Names
With the necessary functions created, we can put everything together to create a function called find_similar() that scans through a list of names and returns a dataframe of results. It takes a list of names as input, as well as a 'similarity threshold'; this is a minimum similarity score that's required in order to be included in the final results dataframe. By doing this, we can reduce the amount of comparisons needed, since we're really only interested in names that are similar and not those that are obviously unrelated to each other (i.e. those with low similarity scores). Then, like the cosine_similarity() function, there's an argument for the number of decimal places to calculate to called ndigits.